DIVERTIRSI CON IL CALCOLATORE

giochi, simulazioni e grafica

a cura di
VIRGINIO SALA

LE SCIENZE S.p.A. EDITORE

DIVERTIRSI CON IL CALCOLATORE

giochi, simulazioni e grafica

a cura di
VIRGINIO SALA

LE SCIENZE S.p.A. EDITORE

Milano, 1987

Il curatore
Virginio Sala si è laureato in filosofia presso l'Università statale di Milano discutendo una tesi di logica matematica. Si è occupato di applicazioni della logica allo studio del linguaggio naturale. Da 15 anni è attivo nel campo editoriale e si è specializzato nella divulgazione scientifica. Ha fatto parte, dal 1977 al 1984, della redazione di «Le Scienze», rivista alla quale ancora collabora. Attualmente è responsabile editoriale di un'importante casa editrice.

Redazione
Elena Bernacchi e Gian Maria Fiameni

In copertina
Ricostruzione al calcolatore dell'insieme di Mandelbrot e delle sue coordinate nel piano complesso ottenuta da Heinz-Otto Peitgen, Peter H. Richter e Dietmar Saupe.

ISBN 88-7004-054-2

Sommario

Introduzione

Il gioco (da tempo psicologi, antropologi, sociologi e filosofi ce lo vanno ripetendo) non è solo un modo piacevole per trascorrere il tempo libero o per ingannare i momenti d'attesa. Per gli adulti non meno che per i bambini, il gioco rappresenta spesso un'attività esplorativa, la possibilità di sondare in condizioni controllate situazioni o porzioni del mondo reale lontane, troppo complesse o su cui comunque la sperimentazione sarebbe impossibile, troppo costosa o troppo pericolosa. Non è un caso, in questa prospettiva, che i bambini nei loro giochi imitino spesso il comportamento dei «grandi»: c'è divertimento in questo, non c'è dubbio, ma è preponderante il tentativo di ripetere (in miniatura) un comportamento, o una classe di comportamenti, per capirli e dominarli.

Anche per il calcolatore (che, si sa, ha un'età ancora molto tenera sulla scala della storia della tecnologia) è avvenuto qualche cosa di analogo: ai giochi (e in particolare a quelli classici come la dama e gli scacchi) si sono interessati un po' tutti i pionieri dell'informatica e i giochi occupano un posto di tutto rilievo nelle ricerche di quel campo che va sotto il nome di Intelligenza Artificiale (crogiolo di idee e punto di incontro di discipline diverse, informatica e psicologia, linguistica e filosofia, che sarebbe fuorviante vedere solo come una branca dell'informatica). Qui il gioco ha proprio quella funzione esplorativa di cui si diceva: situazioni controllate, regole chiare e ben definite, ma anche tutti i problemi della vita reale. Entrano aspetti di tattica e di strategia, comportamenti orientati a un fine, in certi giochi l'informazione incompleta, l'inganno e il bluff. Dal punto di vista della programmazione, ci sono tutti i punti cruciali concettuali, a partire da quello fondamentale della rappresentazione della conoscenza.

John von Neumann ha legato il suo nome alle prime ricerche sui calcolatori elettronici e all'architettura convenzionale di queste macchine, ma anche alla teoria dei giochi (*Theory of Games and Economic Behaviour*, scritto in collaborazione con Oskar Morgenstern, è del 1944); Claude Shannon pubblicò nel 1950 un articolo dal titolo *Programming a Computer for Playing Chess*; anche Alan Turing si interessò agli scacchi (e progettò un programma che però non ebbe modo di realizzare effettivamente). Si può anche risalire più indietro e incontrare i giochi proprio alle radici del calcolatore moderno: persino Charles Babbage aveva pensato a una macchina per giocare a scacchi e a filetto. Babbage aveva escogitato quest'idea come una sorta di mezzo pubblicitario per raccogliere fondi (problema che non lo lasciò mai) e uno spunto analogo fornì qualche incentivo, ben più vicino a noi, per Arthur Samuel, quando si dedicò a un programma per la dama, all'Università dell'Illinois. («Volevamo costruire un calcolatore molto piccolo e tentare di fare con esso qualcosa di spettacolare che richiamasse l'attenzione in modo da poter ottenere dell'altro denaro. Si dava il caso che la primavera successiva ci fosse un campionato mondiale di dama nella vicina cittadina di Kankakee, e così a qualcuno venne l'idea - non sono sicuro che fosse mia, so solo che in seguito ne fui biasimato io - che sarebbe stato bello costruire un piccolo calcolatore che sapesse giocare a dama. Pensavamo che la dama fosse un gioco abbastanza banale. Claude

Shannon aveva parlato di programmare un calcolatore per giocare a scacchi, e altri ci stavano pensando, così decidemmo di prendere un gioco più semplice e di scrivere un programma per giocare a dama. Poi, alla fine del torneo, avremmo sfidato il campione mondiale e lo avremmo battuto, e ciò avrebbe richiamato un mucchio di attenzione. Eravamo molto ingenui.») Fu solo più tardi, però, che Samuel riuscì nel suo intento, dopo il passaggio ai laboratori della International Business Machines, e il suo programma fu tra i primi ad assurgere a notorietà anche presso il pubblico dei non addetti ai lavori.

Sempre nello stesso torno di tempo, anche Alex Bernstein, che prima della guerra era stato un valido giocatore di scacchi, entrò alla IBM e fu attratto dalla sfida di costruire un programma per quel gioco. Il numero delle mosse possibili, negli scacchi, è astronomico (10^{120}, aveva calcolato Shannon nel 1948) e quindi non è possibile pensare a un programma che, per decidere la mossa migliore, le prenda in considerazione tutte e le valuti prima di operare le sue scelte: i procedimenti della ricerca bruta, esaustiva, non hanno spazio qui, come nella maggior parte dei problemi di cui si occupa l'intelligenza artificiale, e Bernstein fu tra i primi a doverne tener conto, e ad avvicinarsi a soluzioni euristiche (empiriche, imperfette) tipiche del nuovo campo di studi.

Le macchine che giocano, e che giocando ci insegnano qualcosa sui modi possibili per costruire calcolatori più intelligenti, da allora hanno fatto molta strada: i programmi che giocano a scacchi sono arrivati a livelli di bravura tali da farli gareggiare senza troppa tema di sfigurare in tornei umani, il programma di Hans Berliner che gioca a backgammon è riuscito a battere il campione mondiale in carica, qualche anno fa, ci sono programmi che giocano a go, a Othello, a Reversi, anche a poker, con risultati tutt'altro che disprezzabili. Ma, parallelamente, è venuto emergendo anche un altro tema che in questi giochi classici trova uno spazio solo molto limitato: quello della simulazione.

«Simulazione» è una parola che, nel linguaggio quotidiano, ha un significato negativo: vale, per lo più, per qualcosa di falso, ingannevole. Nel linguaggio più tecnico della scienza dei calcolatori, invece, ha acquistato un senso decisamente positivo, strettamente legato a quello della parola «modello»: simulare un fenomeno, un'attività o un sistema, significa riprodurne, su diversa scala, le caratteristiche essenziali, per poterli osservare meglio e poter condurre sperimentazioni controllate (dove la «riproduzione su diversa scala» altro non è se non un modello del fenomeno, dell'attività e del sistema). I modelli non sono un'invenzione concettuale del nostro secolo, tutt'altro: sono profondamente radicati in tutta la storia della scienza moderna. Ma il calcolatore ha dato una nuova dimensione all'attività di costruzione di modelli, per fini strettamente scientifici o tecnici e anche per fini di istruzione, addestramento, formazione. Il caso più noto oggi è probabilmente quello dei simulatori di volo: è già rischioso prendere lezioni di scuola guida su un'utilitaria, ma far fare esperienza pratica a un pilota di jet o di mezzi spaziali è ancor più pericoloso (e, anche in condizioni di perfetta sicurezza, comunque molto costoso: e il fattore economico è sempre una molla potente). Ma si può ricostruire tranquillamente a terra l'interno di una cabina di pilotaggio, con tutta la strumentazione opportuna, e si può pensare di racchiudere la cabina in un contenitore a cui si possano inviare sollecitazioni analoghe a quelle che subisce un velivolo. E si può pensare di far controllare il tutto da un calcolatore, che riceva dalla cabina i segnali inviati dagli strumenti e dai dispositivi di guida, e provveda a inviare gli opportuni segnali di ritorno (e magari mostri su un finestrino - uno schermo video - le opportune immagini del paesaggio circostante). Il pilota allora potrà esercitarsi proprio come se si trovasse al suo posto di guida, ma senza alcun rischio reale (e potrà permettersi di schiantarsi senza danni a sé e agli altri). Simulatori di volo più o meno con queste caratteristiche non sono solo un'idea: sono stati effettivamente realizzati e costituiscono uno dei mezzi di addestramento principali, per esempio alla NASA.

Un simulatore di volo ha tutti gli ingredienti di un gioco: potrebbe benissimo essere visto come un esercizio di abilità, di prontezza di riflessi, di competenza, niente affatto privo di fascino. E infatti i simulatori di volo dell'ente spaziale americano hanno avuto i loro nipotini in campo commerciale, decisamente

molto ridotti, ma molto attraenti nonostante la perdita di molte fra quelle caratteristiche che determinano l'utilità dei veri simulatori di volo nei corsi di addestramento.

La simulazione è fondamentalmente un'attività, non una riproduzione statica. Si parte, per esempio, da un sistema reale, che in qualche modo almeno parzialmente si deve riconoscere. Se ne astraggono gli elementi che appaiono pertinenti per quello che si vuole scoprire ulteriormente e se ne costruisce un modello. Così, per esempio, per ottenere una simulazione del moto dei pianeti nel sistema solare, si può pensare (almeno in prima approssimazione) che, date le leggi di Newton, conoscere esattamente la costituzione interna di questi corpi non sia essenziale. Una volta costruito il modello, e opportunamente trasformato in programma per calcolatore, lo si manda in esecuzione e se ne osserva il comportamento, confrontandolo con quello del sistema reale. Le discrepanze possono dare qualche indicazione su come il modello debba essere arricchito e possono dire anche in quale direzione si debba studiare più a fondo il sistema reale. Si può instaurare un ciclo, teoricamente senza fine, tra fasi di arricchimento del modello e fasi di ricerca motivate dalle discrepanze.

Forse ancora più interessante, quando il modello risponde a tutti i requisiti richiesti, è la fase di sperimentazione: non si può realmente provare a spostare Giove per vedere quel che succederebbe del sistema solare, ma in una opportuna simulazione al calcolatore l'esperimento potrebbe essere condotto... quasi senza fatica. E, in fondo, a questo punto potremmo avere anche tutti gli elementi per un gioco di guerre spaziali!

L'intreccio che si può creare fra attività ricreativa, gioco e sperimentazione scientifica potrebbe avere interessanti risvolti in campo educativo: se è vero che il gioco è, nelle sue forme migliori, attività esplorativa del reale, la sua introduzione nelle aule scolastiche e nelle sedi dell'istruzione e della formazione professionale potrebbe avere effetti benefici nel vivificare discipline d'insegnamento complesse e molto astratte. In fondo non è un'idea molto originale: nelle scuole materne e nelle elementari è spesso già applicata con profitto. Il gioco (e il gioco di simulazione in particolare) può costituire una valida alternativa anche nell'istruzione media e superiore, a maggior ragione se veicolato attraverso la potenza del calcolatore. C'è spazio per giocare programmando, per programmare giochi e per imparare giocando.

Con il suo programma per il gioco degli scacchi, Alex Bernstein raggiunse una notorietà insperata: il suo lavoro, fra i primi che possono essere classificati nell'area dell'intelligenza artificiale, non mancò di destare l'interesse della redazione di «Scientific American», che gli chiese la stesura di un articolo (apparso poi nel 1958 con il titolo di *Computer vs. Chessplayer*). Comincia così una serie di pregevoli articoli che hanno dimostrato, nell'arco degli anni, l'attenzione di «Scientific American» per le nuove direzioni di ricerca e la capacità della redazione della rivista americana di intuire la profondità e la validità generale dei problemi chiamati in causa dallo studio dei giochi. Da quasi vent'anni ormai quel materiale viene tempestivamente fornito in versione italiana da «Le Scienze». Dal 1983, poi, quando Martin Gardner ha interrotto la sua rubrica di «Giochi matematici», questi argomenti hanno trovato uno spazio regolare sulle pagine della rivista, prima con i «Temi metamagici» di Douglas Hofstadter (sempre fortemente orientati ai problemi concettuali dell'intelligenza artificiale) e poi con le «(Ri)creazioni al calcolatore», alla cui stesura si sono avvicendati Brian Hayes e A. K. Dewdney.

Quando su «Scientific American» apparvero per la prima volta le «Computer Recreations», e si pose il problema della versione italiana, la decisione non fu facilissima: il titolo che oggi porta la rubrica italiana, con la (Ri) di Ricreazioni posta fra parentesi, non è solo un vezzo, ma intende mettere in evidenza la convinzione delle potenzialità creative (e non solo ludiche), e quindi formative, del gioco al calcolatore. Non c'è bisogno di dire che non si deve equivocare su questa espressione: qui siamo ben lontani dall'apologia dei videogiochi che certa pedagogia di seconda scelta di scuola americana ha voluto fare, intorno all'inizio degli anni ottanta. Se qualcuno ancora nutrisse qualche dubbio, basta che sfogli questo volume e si accorgerà subito che

l'accento è posto sulla partecipazione attiva (e a volte anche impegnativa) del lettore.

La natura di una rubrica di rivista porta spesso alla ripresa di argomenti a distanza di tempo, alla documentazione di un dialogo epistolare con i lettori spesso ricco di spunti, a una certa frammentazione delle informazioni, che poi non è facile ricostruire. Questo volume offre, a quanti sono stati stimolati da Hayes e Dewdney soprattutto, la possibilità di trovare in un'unica sede una scelta ordinata del materiale pubblicato nelle «(Ri)creazioni al calcolatore»: gli articoli sono stati raggruppati secondo alcuni temi principali e in calce a ogni articolo sono stati riportati i brani di ripresa dell'argomento apparsi in numeri successivi della rivista, a seguito delle indicazioni o delle soluzioni inviate dai lettori ai titolari della rubrica.

Virginio Sala

Milano, giugno 1987

I. Giochi per tutti

«Un esame delle macchine che giocano non può non cominciare con un rapido cenno a quella famosa truffa che fu l'automa di Maelzel: costruito nel 1769 dal barone Kempelen, esso giocava a scacchi grazie a un nano nascosto nel suo interno. Ma quanto ci accingiamo a discutere non sono trucchi di questo genere: al contrario, ci occuperemo dei tentativi fatti per costruire macchine che si comportano come se contenessero un uomo, ma che in realtà non lo contengono. Le ricerche che vengono compiute oggi in questa direzione non sfociano sovente nella costruzione di macchine reali, poiché di solito ci si limita a scrivere un programma per un calcolatore digitale già esistente. Quando viene corredato di un particolare programma di gioco, un calcolatore diventa a tutti gli effetti pratici un dispositivo diverso, e può essere definito una macchina che gioca.

«I giochi costituiscono uno strumento di elezione per studiare i metodi con cui simulare certi aspetti del comportamento intelligente: sia perché sono divertenti, sia perché riducono il problema a dimensioni affrontabili. I giochi, specie quelli di antica tradizione, conservano molte caratteristiche essenziali dei problemi della vita reale, mentre sono privi di molte loro fastidiose complicazioni. Si è quindi spinti allo studio delle macchine che giocano da motivi che sono insieme frivoli e seri.»

Così scriveva Arthur L. Samuel nel suo famoso saggio dedicato alla programmazione dei giochi al calcolatore (tradotto in *La filosofia degli automi*, a cura di V. Somenzi e R. Cordeschi, Boringhieri, Torino, 1986, pp. 260-287). Partiamo dunque dai giochi classici, con la dama e gli scacchi: questa prima parte offre poi una carrellata un po' su tutti i tipi di giochi, per arrivare fino al gioco d'avventura e ai giochi spaziali che hanno dominato la fantasia dei progettisti nell'ultimo decennio e più. Una nota particolare merita la terna di articoli dedicati alla Guerra dei nuclei, soggetto portato da Dewdney a notorietà generale proprio dalle colonne di «Scientific American». Il gioco è nelle sue linee generali molto semplice, ma l'invenzione di validi programmi di battaglia è un esercizio non banale, basata com'è (secondo la proposta) su un linguaggio che si avvicina molto al linguaggio d'assemblatore (ecco dunque uno spunto per avvicinarsi piacevolmente ai linguaggi di basso livello!).

L'interesse che la Guerra dei nuclei ha suscitato è testimoniato dalla nascita della Core Wars Society (che pubblica anche un bollettino periodico) e dal buon numero di listati pubblicati sull'argomento dalle riviste di personal computer (in Italia, in particolare, da «Bit» e «MC Microcomputer»). Fra le appendici si trovano le «linee guida» per la costruzione di un programma di Guerra dei nuclei, che in precedenza erano disponibili solo come dattiloscritto.

Un programma che gioca a dama è spesso avanti di un passo

di A. K. Dewdney

Le Scienze, settembre 1984

«Si può scrivere un programma che giochi a dama e che non perda mai con un uomo?». Il ragazzo non aveva più di 10 anni e, seduto al terminale, guardava in su con quell'aria di innocente serietà che solo i giovanissimi sanno avere. Stava giocando a dama contro un programma scritto da alcuni dei miei studenti. Avevano predisposto un livello di difficoltà moderato e il ragazzino stava andando abbastanza bene.

Quella domanda, posta parecchi anni fa in un giorno di visita dell'università, ammette diverse risposte a seconda del tipo di dama a cui ci si riferisce, e tocca anche temi centrali per l'intelligenza artificiale, lo studio del comportamento intelligente in programmi per calcolatori. Tornerò più avanti su questi temi.

Nel momento in cui pose quella domanda, il ragazzo stava giocando una dama 6×6, uno degli infiniti giochi di dama. Qui prenderò in considerazione la dama 4×4, quella 6×6 e quella 8×8 (il gioco classico) e metterò a confronto le prospettive di programmi scritti appositamente per quei giochi. Si può dare una definizione generale della dama di ordine n-esimo su una scacchiera $2n \times 2n$. In ogni caso le pedine occupano tutte le caselle nere tranne quelle delle due righe centrali. Le regole per una dama di ordine n-esimo sono presentate nel riquadro a pagina 13.

Per la dama 4 × 4, la risposta alla domanda del ragazzo è «sì»; per la dama 6 × 6 la risposta è «molto probabilmente» e per quella 8 × 8 «probabilmente». Queste risposte si fondano su una combinazione di esperienza personale, letture e semplici congetture.

Prendiamo in esame il gioco 4 × 4. Ogni parte ha due pedine e raramente una partita supera le 10 mosse, perlomeno quando i giocatori sono esperti. I principianti possono arrivare a 20 o più mosse prima di riconoscere una posizione di parità. A proposito di partite terminate pari, in un torneo di dama 4 × 4 svoltosi a Damenspielspurg furono giocate sei partite patte in fila, mentre la settima andò al gran maestro Samuel Jensen Truscott per un fallo tecnico: il suo avversario, per sbaglio, toccò due volte un pezzo che non si poteva giocare.

A ogni modo, la dama 4 × 4 può essere analizzata con carta e penna. L'analisi dice che è un gioco sempre in parità e un giocatore ragionevolmente attento non perde mai. Lo stesso è vero per un programma ragionevolmente ben scritto.

In una partita di dama 6×6, ogni giocatore ha sei pedine e una partita tipica dura circa 20 mosse. Quando il nostro programma era posto al massimo livello di analisi in profondità, impiegava da cinque a 10 minuti per scegliere una mossa. Su questa posizione, giocava a dama in modo apparentemente invincibile fino alle mosse conclusive. Di solito era così in vantaggio, per numero di pezzi o posizione, da riuscire a chiudere con una vittoria o almeno un pareggio. A volte, però, uno di noi sopravviveva con abbastanza pezzi per batterlo nelle mosse conclusive, quando il suo gioco era sorprendentemente debole.

La dama 8×8 ha 12 pedine per giocatore e le partite tra esperti possono durare 30 mosse o anche più. Attualmente, nei tornei di dama standard, di solito si estraggono a sorte le mosse d'apertura. Questa pratica è stata adottata intorno alla metà del secolo in Inghilterra per la tendenza dei giocatori più esperti a fossilizzarsi sulle proprie aperture favorite e su linee di gioco familiari, con un conseguente aumento delle partite pari. È difficile dire se le partite pari sono comuni perché la dama è di per sé un gioco pari o perché gli esperti hanno un atteggiamento conservatore. La dama 8×8, comunque, rimane un gioco affascinante e difficile; da un punto di vista matematico è un po' meno complesso degli scacchi, ma vi sono esperti in entrambi i giochi che non considerano gli scacchi un gioco «superiore».

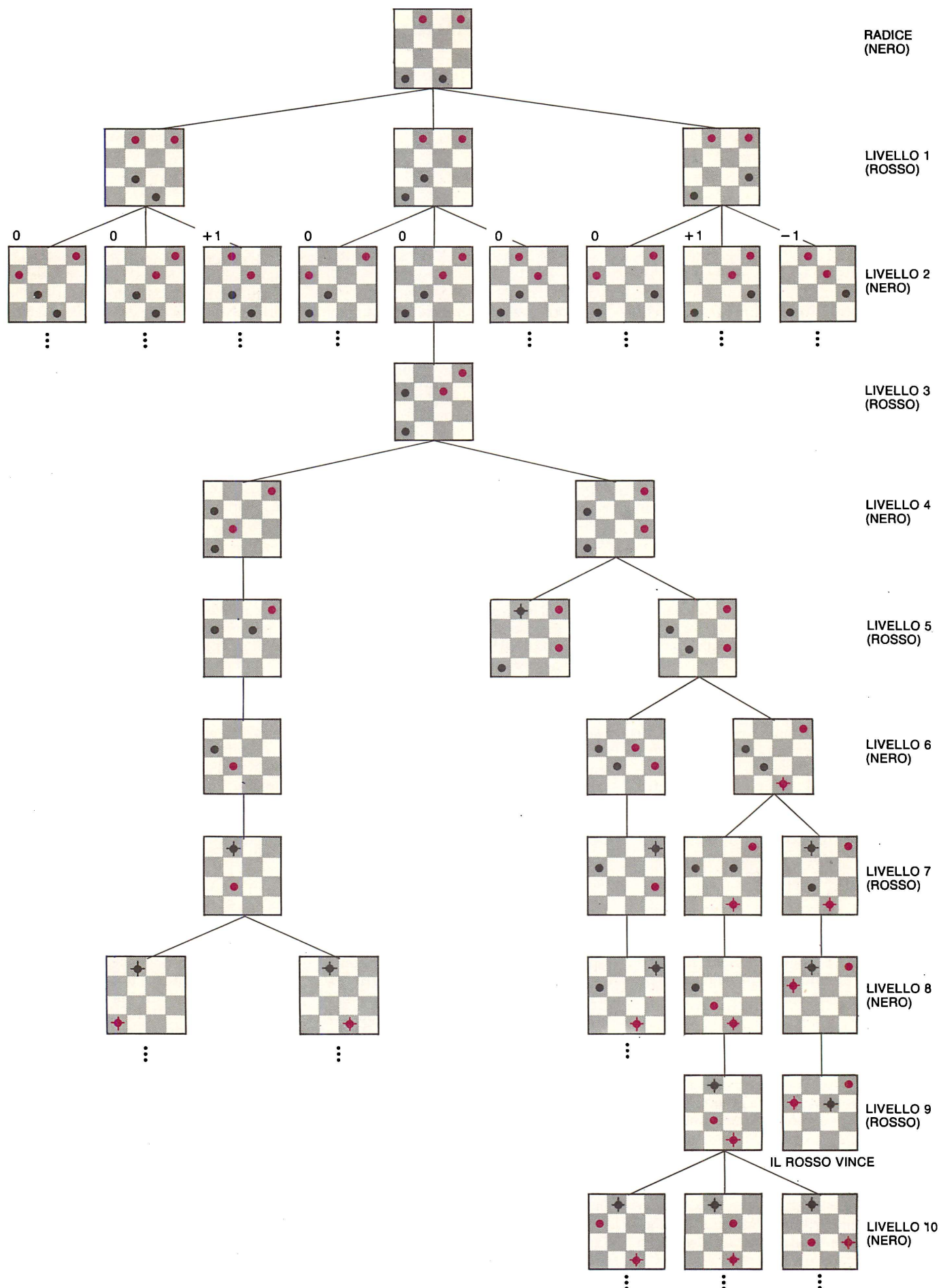
I due programmi per dama più noti sono stati elaborati da Arthur L. Samuel della International Business Machines Corporation negli anni sessanta e da Eric C. Jensen e Tom R. Truscott della Duke University nel 1977. Nel 1962, il programma di Samuel sconfisse Robert Nealey, già campione del Connecticut. Pur essendo molto forte, questo programma non è classificato come esperto. Il programma di Jensen e Truscott è ancora migliore e può essere classificato tra i migliori giocatori del mondo.

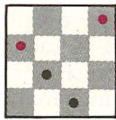
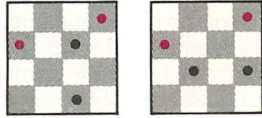
Nel 1979 il programma di Jensen e Truscott giocò contro Elbert Lowder, generalmente noto come uno dei più forti giocatori di dama degli Stati Uniti. Su cinque partite, il programma ne pareggiò due, ne vinse una e ne perse due. Le sconfitte sembravano dovute al gioco conclusivo: quando Lowder si accorse che due delle pedine del programma erano mantenute di guardia sulla riga di fondo, sfruttò questa debolezza e vinse le ultime due partite. Va detto che, giocando contro il programma di Jensen e Truscott, Lowder mantenne un atteggiamento sperimentale e in qualche modo sportivo. La sua unica sconfitta fu chiaramente il risultato di un tentativo di portare il programma su posizioni non ortodosse - una tattica che probabilmente non si dovrebbe scegliere con un programma di questo calibro.

Nel frattempo Marion F. Tinsley, campione mondiale di dama, sta a osservare divertito dalla sua casa di Tallahassee in Florida. Non so se Tinsley abbia mai giocato con uno dei migliori programmi per dama, ma so che i due programmi più forti hanno giocato uno contro l'altro. Nelle due partite contro il programma di Samuel, il programma di Jensen e Truscott si dimostrò, secondo le parole di Truscott, «decisamente superiore» e vinse tutte e due le partite.

Quasi tutti i migliori programmi di gioco hanno tre parti fondamentali: un generatore di mosse, un valutatore di scacchiera e una procedura minimax. Un altro elemento di importanza centrale è l'albero di gioco, che svolge un ruolo in tutti e tre i segmenti del programma. Il generatore di mosse provvede a sviluppare l'albero; il valutatore di scacchiera è consultato alla fine di ogni ramo dell'albero e la procedura minimax viene applicata all'albero nel suo complesso.

Un albero di gioco ha la scacchiera iniziale come radice e ogni ramo rappresenta una posizione che si può raggiungere con una singola mossa. Stranamente, l'albero è di solito disegnato al contrario, con la radice in alto. Può essere distinto in livelli e le scacchiere all'n-esimo livello rappresentano tutte le possibili mosse di un giocatore al turno n-esimo. Pur non essendo di solito molto alti (o profondi), gli alberi di gioco possono essere straordinariamente frondosi; di regola, il numero delle scacchiere a ogni livello è più del doppio del livello precedente. Anche se ogni scacchiera avesse solo due successori, il numero dei rami crescerebbe rapidamente a dismisura: un albero che si biforcasse a questo modo 64 volte avrebbe più rami di tutti gli alberi della Terra. (In natura gli alberi si biforcano poco più di una decina di volte dal tronco alle foglie.) L'esplosione esponenziale del numero dei rami di un albero di gioco fa sì che un calcolatore non possa esaminare più di una piccola frazione delle possibili prosecuzioni in un gioco quale la dama 8×8.

L'albero della dama è creato dal generatore di mosse, un programma che prende in ingresso una particolare distribuzione dei pezzi sulla scacchiera insieme con l'indicazione del giocatore a cui tocca muovere. L'uscita del generatore di mosse è un elenco di tutte le disposizioni sulla scacchiera che si possono raggiungere da quella data con una mossa lecita. Consideriamo la seguente scacchiera 4×4, sup-

L'albero del gioco della dama 4×4, completo fino alla seconda mossa e con alcune prosecuzioni scelte

ponendo che tocchi al Nero muovere:

Il Nero ha due mosse possibili e quindi il generatore di mosse elenca le seguenti due scacchiere:

Il generatore di mosse potrebbe poi essere applicato ulteriormente a queste scacchiere.

I lettori che vogliano studiare come agisce un generatore di mosse sono invitati a sviluppare, partendo da questo livello, l'albero della dama 4×4. Al livello successivo, per esempio, ci sono solo due scacchiere perché in ogni caso le regole costringono il Rosso a mangiare una pedina. Al livello successivo ci sono ancora due scacchiere, poi tre (con una vittoria del Rosso), poi due, poi cinque, poi 10, poi 24. Si consiglia chi volesse andare ancora più avanti di seguire due principi. Primo, controllare a ogni livello se ci sono scacchiere uguali e sviluppare l'albero partendo solo da una di esse. Secondo, partendo da ogni scacchiera controllare in alto se la posizione si è già presentata; se così è, chiudere quel ramo e segnarlo «pari».

Come funziona un programma per generare mosse? In primo luogo bisogna vedere come viene rappresentata una scacchiera. Lo schema più semplice è una matrice, una tabella a due dimensioni, i cui elementi identificano la posizione delle pedine. Ad esempio, una casella vuota potrebbe essere rappresentata da uno 0, una casella con una pedina da un 1 e una casella con una dama da un 2; il segno dell'elemento denoterebbe il colore, per esempio più per il Nero e meno per il Rosso. La prima scacchiera 4×4 riprodotta sopra avrebbe questa codificazione nella matrice:

$$\begin{matrix} 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & 0 & +1 & 0 \end{matrix}$$

Le mosse lecite per ogni pedina dipendono dal suo colore, dalla sua posizione e dal fatto che le caselle diagonalmente adiacenti siano o meno occupate. In questo caso il generatore di mosse controlla la matrice riga per riga finché arriva all'elemento positivo (Nero) della riga 3 e colonna 2, una posizione indicata (3,2). Dato che tocca al Nero muovere e che si tratta di una pedina (grandezza 1) e non di una dama, il generatore di mosse esamina le caselle (2,1) e (2,3). La casella (2,1) è già occupata da una pedina (e non si può saltarla), ma la (2,3) è libera. Di conseguenza, il programma scrive una nuova matrice con il +1 tolto dalla casella (3,2) e posto in (2,3):

$$\begin{matrix} 0 & 0 & 0 & -1 \\ -1 & 0 & +1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 \end{matrix}$$

Il generatore di mosse continua poi il suo controllo della matrice finché trova il successivo elemento positivo.

Per riuscire a tener conto delle scacchiere e delle corrispondenti matrici, si possono numerare a mano a mano che vengono generate. Qualsiasi casella di qualsiasi scacchiera può poi essere identificata dal numero di scacchiera e dalla riga e dalla colonna. Quando il generatore di mosse costruisce l'albero della dama, deve conservare una documentazione dei rapporti fra le scacchiere. Nella scelta di una mossa, il programma esplora l'albero di gioco scendendo di parecchi livelli, poi risalendo, poi scendendo di nuovo e così via. Per attraversare l'albero così, il programma deve poter riconoscere il «genitore» di una scacchiera e i suoi «figli». È dotato, a questo fine, di una «tabella dei puntatori»: per ogni scacchiera, il puntatore è un elenco di numeri, il primo dei quali indica il genitore e i successivi i suoi figli.

Il successivo componente fondamentale del programma per la dama, il valutatore di scacchiera, riceve in ingresso una scacchiera e calcola un numero che rispecchia il valore della scacchiera per il Nero; un numero alto significa che il Nero probabilmente vincerà se nel gioco reale si raggiunge quella posizione. La difficoltà nel progettare un buon valutatore di scacchiera sta nel fatto che sono molti i fattori da prendere in considerazione per giudicare una posizione e non è sempre chiaro che peso si debba dare ai vari fattori. Quello che segue è un breve elenco di fattori relativamente primitivi.

PEDINE: aggiungere 1 per ogni pedina Nera sulla scacchiera; sottrarre 1 per ogni pedina Rossa.

DAME: aggiungere 1 per ogni dama Nera; sottrarre 1 per ogni dama Rossa.

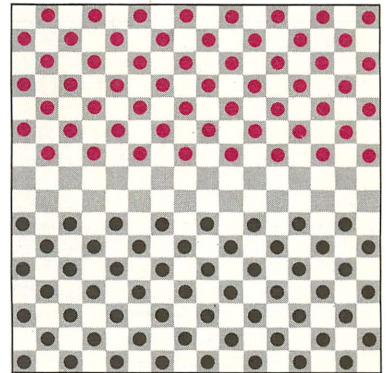
CENTRO: aggiungere 1 per ogni pedina Nera che occupa una delle quattro caselle centrali; sottrarre 1 per ogni pedina Rossa che occupa quella posizione.

AVANZAMENTO: aggiungere 1 per ogni pedina Nera sulla seconda riga, 2 per ogni pedina Nera sulla terza riga, 3 per ogni pedina Nera (dama) sulla quarta riga. Sottrarre i numeri corrispondenti per il Rosso.

MOBILITÀ: aggiungere 1 per ogni mossa possibile per il Nero e sottrarre 1 per ogni mossa possibile per il Rosso.

Qual è l'importanza relativa di questi fattori? Le DAME devono avere un peso doppio rispetto alle PEDINE? L'AVANZAMENTO deve includere già l'informazione relativa alle DAME e alle PEDINE?

Nella soluzione di Samuel a questi problemi, il programma stabiliva che peso dare a ciascun fattore basandosi sulla propria esperienza di gioco. Il valutatore di scacchiera progettato da Samuel dava inizialmente lo stesso peso a tutti e 39 i fattori presi in considerazione (molti suggeriti da esperti di dama e da manuali) e sommava semplicemente i numeri per

Le regole della dama di ordine n-esimo

Su una scacchiera $2n \times 2n$ si alternano caselle chiare e caselle scure e ogni giocatore mette n^2-n pedine sulle caselle scure ai lati opposti della scacchiera. Le due righe centrali sono vuote. Ogni giocatore ha un «angolo singolo» (una casella scura che occupa un angolo) a sinistra; l'altro angolo è detto angolo doppio. Il Nero muove per primo.

Una pedina può muovere in avanti in diagonale su una casella adiacente non occupata. Se quella casella è occupata da un pezzo nemico ma la casella successiva nella diagonale non è occupata, la pedina può saltare il pezzo e toglierlo dalla scacchiera. Tutte le volte che si ha la possibilità di «mangiare» bisogna farlo; se si possono mangiare due o più pezzi, bisogna mangiarne almeno uno. Se, dopo aver mangiato un pezzo, una pedina può mangiarne un altro, deve farlo.

Quando una pedina avanza fino all'ultima riga del lato opposto della scacchiera, si trasforma in una dama. Una dama può muovere sia avanti sia indietro (sempre però in diagonale) e può mangiare in entrambe le direzioni.

Il gioco termina con la vittoria di un giocatore quando l'altro giocatore non può più fare una mossa lecita perché tutti i suoi pezzi sono stati mangiati oppure immobilizzati. Se entrambi i giocatori ritengono che una vittoria sia impossibile, la partita viene dichiarata pari.

Vi sono altre regole a proposito del giocherellare con i pezzi, fumare e imprecare, ma difficilmente un calcolatore le trasgredisce, quindi le ometteremo. Il disegno in alto mostra la posizione di partenza della dama di ordine otto, giocata su una scacchiera 16×16 con 56 pezzi per ciascun giocatore.

ottenere il valore di una scacchiera. Ogni partita perduta veniva analizzata dal programma e il fattore che aveva portato alla mossa perdente veniva isolato e il suo peso ridotto. Il processo veniva accelerato automatizzando l'analisi e facendo giocare il programma con se stesso per un certo numero di volte.

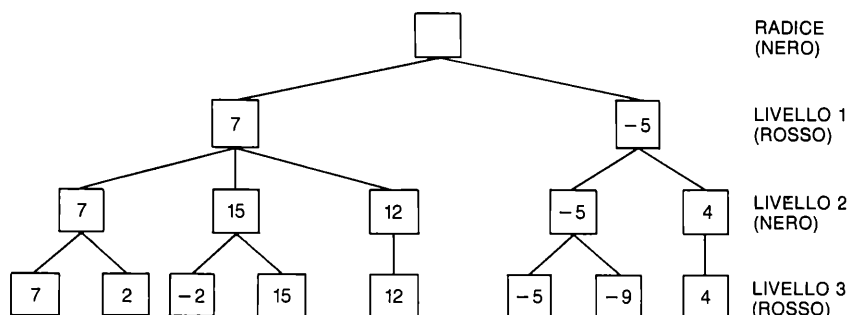
Le procedure per la valutazione delle scacchiere non sono mai perfette. Se lo fossero, gli altri componenti di un programma di gioco non sarebbero necessari: basterebbe che il programma esaminasse le scacchiere di una mossa avanti, le valutasse una per una e poi scegliesse la mossa migliore. C'è una sorta di compensazione tra la precisione del valutatore di scacchiera e il numero di livelli successivi che il programma deve considerare. Se il valutatore non potesse fare molto più che riconoscere una posizione vincente o perdente, l'albero della dama dovrebbe essere esaminato tutto fino in fondo!

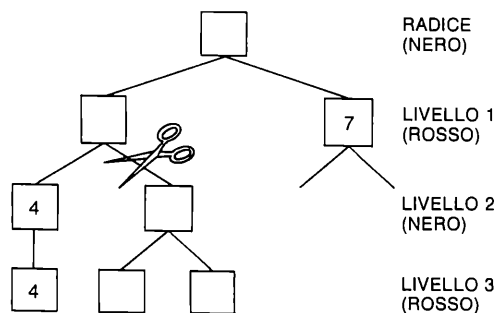
Il terzo componente fondamentale è la procedura minimax. Si tratta, in essenza, di un programma di esplorazione dell'albero che utilizza i valori di scacchiera del più basso livello esaminato per assegnare valori alle scacchiere più in su nell'albero. A qualche punto dell'esplorazione tutte le scacchiere di un certo livello devono essere classificate dal valutatore di scacchiera. A una scacchiera *B* del livello subito superiore viene attribuito un valore secondo questa semplice regola: se in *B* tocca al Rosso muovere, scegliere come valore di *B* il valore minimo trovato tra i figli di *B*; se tocca al Nero muovere, scegliere il valore massimo.

Dovrebbe essere chiaro qual è il ragionamento su cui si fonda questa regola. Se tocca al Rosso muovere, il Nero può presupporre che il Rosso sceglierà una mossa che minimizzi per il Nero il valore della successiva scacchiera. Per la stessa ragione, se è il turno del Nero verrà scelto il valore massimo. La procedura minimax, quindi, parte dal più basso dei livelli dell'albero che si stanno esaminando a un certo momento, valuta tutte le scacchiere a quel livello e fa poi risalire quei valori lungo l'albero. Passando da un livello a quello immediatamente superiore, alternativamente minimizza e massimizza, a seconda del turno di gioco. Infine arriva alla scacchiera attualmente in gioco e fornisce al Nero un valore per ogni possibile mossa. Naturalmente il Nero sceglie la mossa con il valore più alto.

In un programma per la dama non ci sono solo questi tre componenti. Deve interagire con un essere umano, accetta indicazioni di mosse da una tastiera e stampa le mosse che decide di fare. Inoltre può visualizzare un disegno della scacchiera con simboli che rappresentano i due giocatori e i loro pezzi. In molti casi il programma consente anche al giocatore umano di scegliere a quale «livello di abilità» debba giocare il programma. Il livello è di solito un numero che rispecchia la capacità di previsione del programma, cioè il numero di livelli che deve esplorare sull'albero della dama a partire dalla posizione data.

Quando la capacità di previsione è regolata su un livello piuttosto profondo, il programma impiega più tempo; ogni livello in più da esplorare nell'albero può raddoppiare il tempo necessario. Per ridurre la quantità di ricerca che il programma deve compiere, nella procedura minimax può essere incluso un ingegnoso espediente detto potatura alfa-beta e illustrato nel riquadro qui a fianco. La potatura alfa-beta può essere effettuata pratica-

La procedura minimax e la potatura dell'albero

La procedura minimax si applica a un albero di gioco dopo che sono state generate tutte le mosse lecite fino a qualche profondità prestabilita e il valutatore di scacchiera ha stimato il valore della posizione raggiunta alla fine di ogni ramo. Qui sta al Nero giocare, l'albero è stato esplorato fino al terzo livello e il valutatore di scacchiera ha assegnato a ogni scacchiera terminale un intero che rispecchia il suo valore relativo per il Nero. Il compito della procedura minimax è quello di scegliere una mossa per il Nero che si basi su questi valori.

Si noti che la sequenza di mosse più favorevoli per il Nero passa attraverso le caselle che portano i numeri 7, 15 e 15. Il Rosso, da parte sua, favorirebbe le mosse che conducono alla successione −5, −5 e −9. La procedura minimax tiene conto di queste preferenze in conflitto. Il programma esamina dapprima tutte le mosse al livello 3 che sono «figlie» di una data mossa «genitrice» al livello 2. Dato che al livello 2 tocca al Nero muovere, al genitore è assegnato il valore del figlio con il valore più alto. La stessa procedura è seguita per assegnare valori alle scacchiere del livello 1, ma in questo caso si scelgono i valori più bassi perché tocca al Rosso muovere. Se entrambi i giocatori valutano le scacchiere allo stesso modo, il gioco seguirebbe un percorso lungo le mosse con valori 7, 7 e 7.

La ricerca lungo alberi per più di pochi livelli può rivelarsi un compito spropositato e solo potando gli alberi stessi si può effettuare una ricerca completa. Qui tocca al Nero muovere e una ricerca parziale ha già dato il valore 7 a una delle scelte del Nero a livello 1. Esaminando l'altra scelta, il programma ha trovato che una delle possibili risposte del Rosso al livello 2 ha il valore 4. Ne segue che non c'è alcuna ragione di esplorare il resto dell'albero; dato che il Rosso sceglierà la mossa di valore più basso, la mossa alternativa del Nero al livello 1 non può avere un valore maggiore di 4 e quindi è preferibile la mossa a cui è assegnato il valore 7. L'eliminazione di percorsi che il Nero può ignorare è detta potatura alfa; il corrispondente processo per il Rosso è la potatura beta.

mente a tutti i livelli e può eliminare un'enorme quantità di ricerca e di valutazione non necessarie.

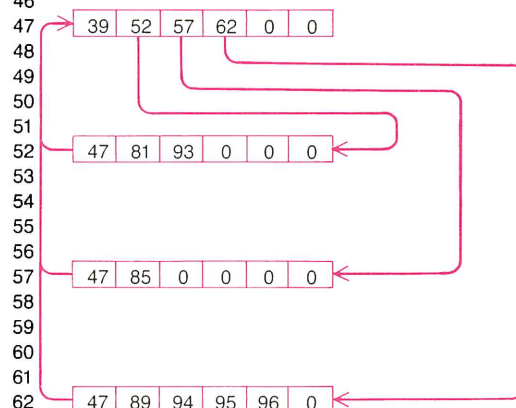
Non è difficile programmare l'algoritmo per il gioco della dama. Truscott ritiene che non sia necessario disporre di un grande calcolatore *mainframe* per giocare in modo efficace. «Il microelaboratore va benissimo per sviluppare programmi che giocano a dama - scrive Truscott - e gli appassionati di ingegno daranno probabilmente significativi contributi a quest'area stimolante, ricca di soddisfazioni e ancora del tutto aperta.»

I lettori con qualche esperienza di programmazione troveranno le indicazioni date in precedenza abbastanza complete per progettare e scrivere un programma che giochi a dama. Altri lettori sono forse ancora incerti sul modo in cui procedere. La costruzione di un generatore di mosse e di un valutatore di scacchiera è ragionevolmente semplice, ma la procedura minimax può richiedere un esame più dettagliato. Come fa un programma a muoversi su e giù lungo l'albero di gioco, ricordando dove è già passato e decidendo dove andare al passo successivo? Nel riquadro di questa pagina è dato qualche consiglio sul modo di muoversi lungo un albero.

Una volta che un programma di gioco è stato progettato, scritto e messo a punto, di solito i suoi limiti appaiono evidenti quando gira con un valore elevato della capacità di previsione. Le mosse possono richiedere troppo tempo oppure la quantità di spazio necessaria per immagazzinare l'albero di gioco può eccedere la disponibilità di memoria. Ci sono molti trucchi per ridurre le esigenze di tempo e di spazio. Per esempio: ho rappresentato ogni scacchiera con una matrice bidimensionale in cui metà degli elementi (quelli corrispondenti alle caselle non significative) sono sempre zero. La scacchiera può essere ridotta a una matrice a una dimensione eliminando gli spazi vuoti; questo, però, rende più complessa la regola per generare le mosse.

La domanda se sia o meno possibile scrivere un programma per la dama che non perda mai con un essere umano si riduce a una domanda sulla natura della dama stessa. Esiste, nella dama di ordine n-esimo, una strategia con la quale il Nero possa sempre vincere? Può il Rosso arrivare a vincere sempre? La dama è un gioco che finisce sempre in parità? È possibile un'analisi con carta e penna della dama di ordine n-esimo solo quando n è uguale a 2. Io ho analizzato la dama 4×4 tracciando l'albero di gioco fino a una media di 10 mosse. A questo livello ci sono poche vittorie per il Nero e un numero altrettanto basso di sconfitte; le altre posizioni sembrano tutte posizioni di parità in cui le dame si rincorrono a vicenda lungo i bordi della scacchiera 4×4. Non ho trovato la conferma della parità in tutti i casi, ma, se le mie valutazioni sono corrette, la procedura minimax dà alla scacchiera di apertura valore zero: una parità.

La dama 6×6 è un gioco che termina

Attraverso l'albero di gioco

Quando un programma per la dama deve scegliere una mossa, deve esplorare l'albero di gioco fino a qualche profondità prestabilita, visitando ogni ramo esattamente una volta. Un metodo semplice per attraversare l'albero utilizza una tabella di puntatori, che registra le relazioni tra rami, e una struttura di dati detta catasta (o pila) *pushdown* che controlla la posizione attuale del programma nell'albero e conserva un elenco dei rami ancora da visitare.

Quando viene generata una nuova scacchiera le viene assegnato un numero, che serve da indice alla matrice di puntatori. Il puntatore associato a una scacchiera è a sua volta un elenco di numeri di scacchiera. Il primo numero dell'elenco è quello del genitore della scacchiera; tutti gli altri numeri indicano i suoi figli. Se il programma ha raggiunto la scacchiera 47, il contenuto del puntatore dice che si dovrebbe proseguire la ricerca con le scacchiere 52, 57 e 62. Se il programma è alla scacchiera 62 e deve ritornare al genitore, il primo elemento del puntatore lo dirige indietro verso la scacchiera 47.

È compito della catasta pushdown seguire il progresso del programma lungo la tabella di puntatori. La catasta stessa può essere creata con una variabile CATASTA e una variabile detta CIMA, che contiene l'indirizzo dell'elemento in cima alla catasta. Ogni volta che un elemento è aggiunto alla catasta (operazione chiamata *push*) o tolto da essa (operazione *pop*), CIMA viene rispettivamente aumentato o diminuito di uno.

L'elemento in cima alla catasta è la successiva scacchiera da esaminare. Se 47 è l'elemento in cima, l'algoritmo di attraversamento dell'albero toglie quel valore dalla catasta ed esamina poi l'elenco dei contenuti del puntatore per la scacchiera 47. Il primo numero dell'elenco (che si riferisce al genitore della scacchiera 47) viene ignorato, ma gli altri elementi non nulli vengono impilati sulla catasta e il valore di CIMA è modificato di conseguenza. Viene quindi scelta la scacchiera 62 per la successiva ricerca.

Ogni volta che il programma raggiunge una scacchiera alla massima profondità consentita, viene chiamato in azione il valutatore di scacchiera, che deve classificarla. La procedura minimax può poi consultare l'elenco dei contenuti del puntatore per trovare il genitore della scacchiera e dargli un valore minimo o massimo provvisorio. Con successive visite a scacchiere sorelle la procedura avrà la possibilità di rivedere quel valore.

sempre pari? Quando il programma scritto dai miei studenti giocava contro se stesso, con una capacità di previsione elevata, raramente le partite arrivavano a una conclusione. Basandosi solo su questo, si potrebbe essere tentati di avanzare la congettura che la dama 6×6 sia un gioco pari. Naturalmente, se la dama 4×4 e la dama 6×6 risultassero entrambe pari, l'ombra del sospetto cadrebbe pesantemente sulla dama 8×8. Forse qualcuno, leggendo quanto sopra, avrà quella rara combinazione di abilità e audacia (o folle temerarietà?) necessaria per scrivere un analizzatore dell'albero della dama 6×6. Su un grande calcolatore il progetto può essere a malapena realizzabile.

A proposito della dama di ordine n-esimo, i teorici della computazione sono arrivati a risultati interessanti. Risulta che il seguente problema sia computazionalmente intrattabile: «Data una qualsiasi posizione lecita delle dame e delle pedine del Nero e del Rosso sulla scacchiera $2n \times 2n$, decidere se il Nero può vincere». Nel 1978 venne dimostrato da Ariezri Fraenkel del Weizmann Institute of Science in Israele e da Michael R. Garey e David Johnson degli AT&T Bell Laboratories che questo problema è «P-arduo rispetto allo spazio» *(P-space hard)*. Il che è ancora peggio che essere NP-completo e la maggior parte dei teorici ritengono che con tutta probabilità i problemi classificati come NP-completi non hanno alcuna soluzione generale pratica. Questa scoperta fa pensare che se un programma sapesse giocare in modo perfetto a dama di ordine n-esimo, il tempo medio per generare una mossa crescerebbe più velocemente di qualsiasi funzione polinomiale di n.

La possibilità che un programma non perda mai una partita di dama dipende molto dalla potenza di calcolo. Anche se la dama fosse in sé un gioco pari, un programma efficace avrebbe indubbiamente un fattore di previsione abbastanza ampio, a parità degli altri elementi. Scrive Truscott a proposito del programma da lui sviluppato insieme a Jensen: «A una media di cinque secondi per mossa (su un IBM 370/165), il programma è in grado di battere i suoi autori. A una media di 20 secondi per mossa, il programma è forse tra i primi 100 giocatori di dama degli Stati Uniti. A una media di 80 secondi per mossa, il programma è forse il decimo giocatore del mondo».

A volte è deludente leggere la descrizione di un programma di gioco in cui siano messe a nudo le sue operazioni interne. È abbastanza facile, quando si gioca contro un programma e non si sa nulla del suo funzionamento, attribuirgli meravigliosi poteri intellettuali che semplicemente non ha. Molti traggono piacere dalla fantasia. Si può solo sperare che la disillusione provocata dalla sua perdita sia compensata dal piacere di osservare la struttura e il modo d'operare dei programmi di gioco.

Il ragazzo aveva chiesto se è possibile scrivere un programma per dama che non perda mai con un essere umano. La domanda tocca due temi fondamentali per l'intelligenza artificiale: che cos'*è* l'intelligenza e fino a che punto si può far sì che i programmi si comportino con intelligenza? Quando si focalizzano queste domande su forme ristrette di attività umana «intelligente» come il giocare, il problema sembra dissolversi in una massa di dettagli tecnici. Sta forse qui la risposta?

La ricerca di una teoria dell'intelligenza continua. Alcuni studiosi di intelligenza artificiale hanno suggerito che costruire una macchina che pensa sia in qualche modo analogo a costruire una macchina che vola. È forse possibile una teoria della «dinamica del volo intellettuale». Per volare non è necessario costruire un uccello meccanico: va benissimo un aeroplano. Sarà mai possibile realizzare qualche cosa del genere per l'intelligenza artificiale?

Yin e yang: ricorsività e iterazione, la Torre di Hanoi e gli Anelli cinesi

di A. K. Dewdney

Le Scienze, gennaio 1985

I buoni rompicapo sono un'ottima guida per addentrarsi nel regno del pensiero astratto abitato da matematici e altri teorici. I migliori rompicapo incorporano temi di questo regno, il cui significato va ben al di là dei rompicapo stessi.

Due rompicapo classici, la Torre di Hanoi e gli Anelli cinesi, richiamano due coppie di argomenti contrastanti: ricorsività e iterazione, unità e diversità. A parte queste considerazioni serie, i rompicapo sono divertenti e danno anche al neofita un gratificante senso di confusione, garanzia del suo lento ingresso nel regno del pensiero astratto.

Il rompicapo della torre è costituito da tre perni verticali posti su una tavola. Un certo numero di dischi, di dimensione crescente, sono all'inizio accatastati su un perno in modo che più in alto di tutti ci sia il disco più piccolo. I dischi vengono mossi secondo le seguenti regole:

1. Si deve spostare un solo disco alla volta da un perno all'altro.
2. Nessun disco può essere messo su un disco più piccolo.

Alla prima mossa si deve spostare il disco più piccolo, dato che è l'unico accessibile all'inizio *(si veda la figura in basso)*. Al turno successivo ci sono due mosse per il disco più piccolo (entrambe inutili) e una mossa per il disco appena più grosso. Questo va sul perno vuoto, visto che non può essere messo sopra il disco più piccolo (regola 2). Al terzo turno quello che si deve fare non è più così ovvio: si deve riportare il secondo disco sul perno iniziale o si deve muovere di nuovo il primo disco e, in questo caso, su quale perno?

Da questo punto in avanti ci si trova di fronte a una lunga successione di mosse e sono numerose le occasioni di effettuare scelte sbagliate. Anche effettuando tutte le mosse giuste, sono necessarie 2^{n-1} mosse (come vedremo in seguito) per trasferire una torre di n dischi, uno alla volta, su un altro perno. Il seguente racconto, tratto dal classico libro di W. W. Rouse Ball *Mathematical Recreations and Essays,* dà una buona illustrazione dell'incredibile quantità di tempo necessaria per risolvere un rompicapo fatto anche di un modesto numero di dischi:

«Nel grande tempio di Benares... sotto la volta che segna il centro del mondo, vi è un piatto d'ottone su cui sono fissati tre perni di diamante, ciascuno alto un cubito e dello spessore del corpo di un'ape. Su uno di questi perni, al momento della creazione, Dio mise sessantaquattro dischi d'oro puro, il più grande appoggiato al disco d'ottone e via via gli altri sempre più piccoli. È la Torre di Brahma. Giorno e notte, incessantemente, i sacerdoti trasferiscono i dischi da un perno di diamante a un altro secondo le fisse e immutabili leggi di Brahma, le quali richiedono che il sacerdote al lavoro non muova più di un disco alla volta e che egli debba porre questo disco su un perno in modo che sotto di esso non vi sia un disco più piccolo. Quando i sessantaquattro dischi saranno trasferiti in questo modo dal perno su cui Dio li aveva messi al tempo della creazione a uno degli altri perni, torre, tempio e bramini si ridurranno in polvere e il mondo svanirà in un colpo di tuono.»

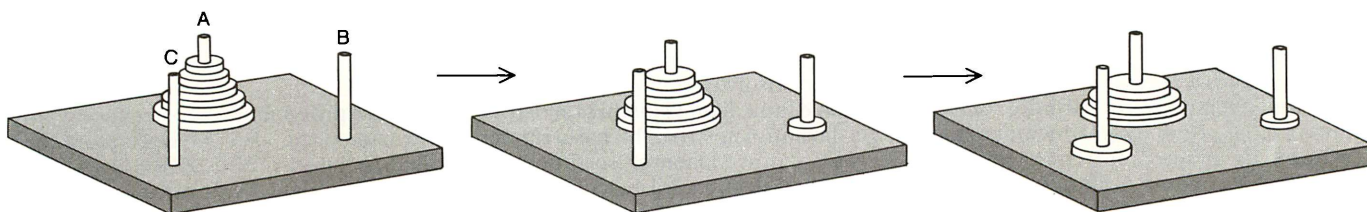
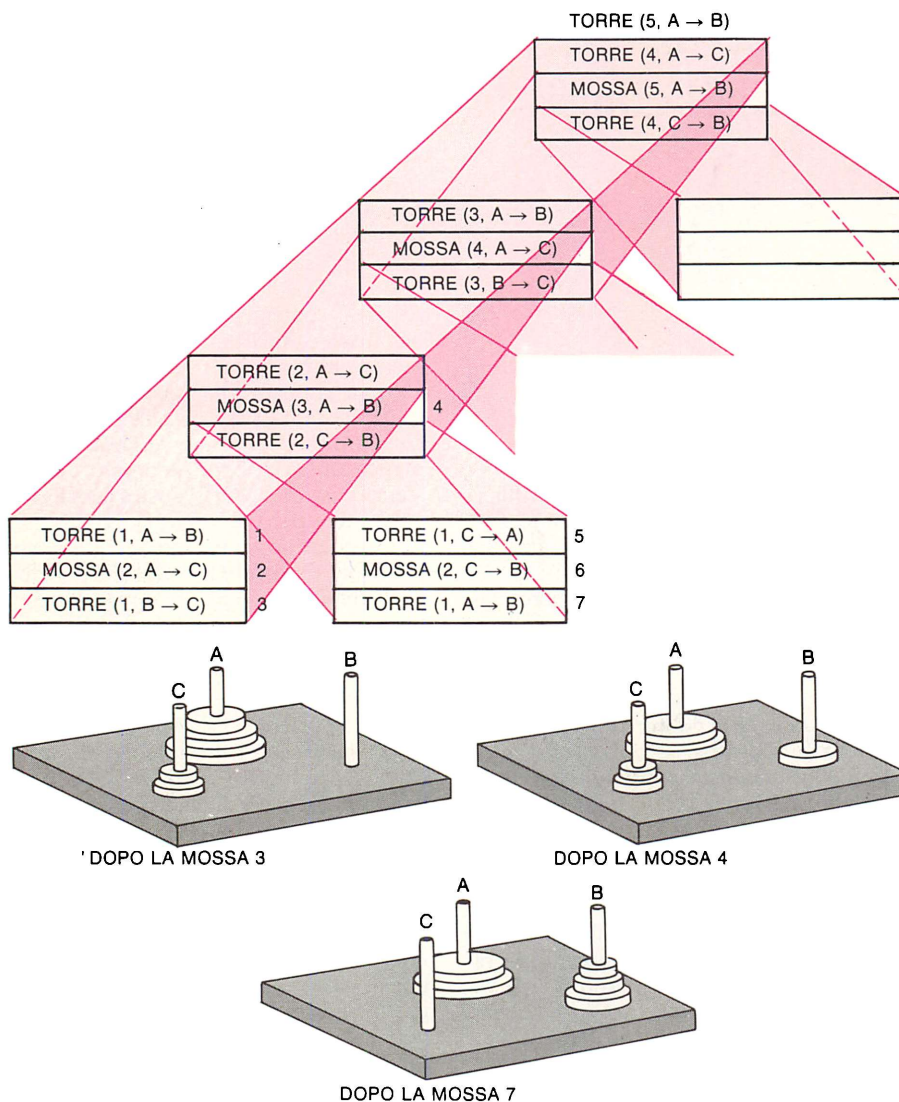
Il fatto che il mondo non sia ancora svanito attesta l'estrema lunghezza del tempo necessario per risolvere il rompicapo: anche muovendo un disco al secondo, i sacerdoti impiegherebbero più di 500 miliardi di anni a trasferire la torre iniziale di 64 dischi!

A questo punto (e senza rischio per l'universo) il lettore può impegnarsi più direttamente prendendo cinque carte da gioco, per esempio dall'asso al cinque di cuori, e visualizzando tre punti su un tavolo. Messe in pila le carte su uno dei punti, in ordine, in modo che l'asso sia in alto, si può tentare di risolvere il rompicapo della torre a cinque dischi spostando una carta alla volta tra due punti - ma senza mai sovrapporre una carta a un'altra di valore minore. Siete in grado di completare il trasferimento della torre di cinque carte prima della fine del mondo? Secondo la formula 2^{5-1}, il trasferimento dovrebbe essere possibile in 31 mosse.

Come si risolve un rompicapo di questo genere? Come avviene che alcune persone trovino facile risolvere rompicapo mentre altre devono fare grandi sforzi? La mia risposta alla seconda domanda suggerisce una risposta alla prima: sono convinto che tutti usino il pensiero matematico quasi in ogni momento dell'esistenza cosciente. Sia le nostre conclusioni sul motivo per cui lo zio Harry non si è fatto vedere al matrimonio, sia il nostro piano per sistemare le valige nel portabagagli della macchina derivano logicamente da certe premesse. Queste deduzioni possono essere molto raffinate, il che mi spinge a ritenere che quasi tutti quelli che sono in grado di compiere simili prodezze intuitive potrebbero diventare buoni pensatori analitici. Il trucco sta nel portare le capacità analitiche intuitive a livello di consapevolezza, affinché possano essere utilizzate in modo formale.

Per esempio, dopo aver giocato per un po' con la torre a cinque dischi, si sarà certamente notato che di tanto in tanto tendono ad apparire torri più piccole. Abbastanza spesso si incontrano torri a due dischi, a volte torri a tre dischi e forse perfino una torre a quattro dischi. Questo può avvenire anche quando non si ha un'idea chiara della soluzione. Si sta semplicemente giocando.

Abbastanza presto, però, emerge un'idea chiave, che può essere sottoposta a un esame cosciente: «Se si può trasferire una torre di due dischi o una torre di tre dischi (per non parlare di una torre di quattro dischi), perché non si dovrebbe poter trasferire una torre di cinque dischi?» Individuando certe regolarità nel modo in cui si formano queste torri più piccole, si arriva, più o meno rapidamente, allo schema di mosse che portano una torre di cinque dischi su uno degli altri perni.

Un'idea analoga si presenta, nella matematica e nella scienza dei calcolatori, come una tecnica per la soluzione di problemi. Posta sotto forma di principio: «Se riesco a risolvere il problema in un caso in qualche modo più piccolo di quello che ho di fronte, forse posso usare questa soluzione nel caso più grande». Questa è la nozione di ricorsività - l'inclusione, in un

Le prime due mosse del rompicapo della Torre di Hanoi

Soluzione ricorsiva del rompicapo della Torre di Hanoi

procedimento, del procedimento stesso.

L'idea di ricorsività, applicata al rompicapo della torre, è molto chiara. Se possiamo risolvere un rompicapo della torre per n-1 dischi, allora possiamo sicuramente risolverne uno per n dischi. Lo spunto principale per sviluppare una soluzione con n dischi da una soluzione con (n-1) dischi viene dalla soluzione della versione con due dischi. Supponiamo che due dischi - il disco in alto, o primo, e il disco in basso, o secondo - debbano essere spostati su un altro perno. Chiamiamo perno sorgente quello che occupano attualmente, perno bersaglio quello che occuperanno alla fine e perno di supporto quello restante. Se spostiamo il primo disco sul perno di supporto e il secondo disco sul perno bersaglio, alla terza mossa il primo disco viene messo sul perno bersaglio, completando così la soluzione. Queste tre mosse diventano la base di una soluzione ricorsiva in tre parti del rompicapo in cui il primo disco è mentalmente sostituito da un'intera torre fatta con n-1 dischi e il secondo è sostituito dal disco n-simo (e più grande). I tre stadi possono essere rappresentati in questo modo:

1. Trasferire la torre di n-1 dischi dal perno sorgente al perno di supporto.
2. Spostare il disco n-simo dal perno sorgente al perno bersaglio.
3. Trasferire di nuovo la torre di (n-1) dischi, questa volta dal perno di supporto al perno bersaglio.

Questa ricetta in tre parti imita soltanto la nostra soluzione del rompicapo con due dischi. Supponendo, naturalmente, che siamo in grado di risolvere il rompicapo con (n-1) dischi, usiamo la successione di mosse della soluzione per trasferire la torre di n-1 dischi dal perno sorgente al perno di supporto. Allo stadio successivo, l'n-simo disco è stato liberato del fardello sovrastante e possiamo spostarlo sul perno bersaglio. Al terzo stadio, applichiamo di nuovo la nostra soluzione per il caso di n-1 dischi per trasferirli dal perno di supporto al perno bersaglio.

Come si risolve il rompicapo della torre quando è fatta di (n-1) dischi? La risposta è già davanti ai nostri occhi: basta ripetere lo stesso procedimento, sostituendo il termine n-1 degli stadi 1 e 3 con n-2 e così via. Alla fine arriviamo al punto in cui gli stadi 1 e 3 richiedono il trasferimento di un unico disco. Questo processo risolutivo, con le sue ramificazioni che si propagano incessantemente, è quanto mai sconcertante per un essere umano, ma chiarisce il motivo per il quale la soluzione ha quasi 2^n passi: ogni volta che viene utilizzato, il procedimento si ripete due volte di più. Se gli uomini trovano difficile realizzare un processo risolutivo di questo tipo, i calcolatori certamente no.

La figura di questa pagina fornisce una parte della soluzione per una torre con cinque dischi. Tre stadi appaiono come procedimenti inclusi in un procedimento più esteso detto TORRE *(n, X → Y)*. Questo procedimento utilizza tre elementi di informazione: *n*, il numero di dischi da muovere; *X*, il perno sorgente, e *Y*, il perno bersaglio.

Come suggerisce la terminologia utilizzata, lo schema essenziale di un programma ricorsivo per il rompicapo della torre può avere la seguente formulazione algoritmica:

TORRE (n, *sorgente → bersaglio):*
  TORRE (n-1, sorgente → supporto)
  MOSSA (n, sorgente → bersaglio)
  TORRE (n-1, supporto → bersaglio)

Supponiamo di voler spostare cinque dischi dal perno *A* (la sorgente) al perno *B* (il bersaglio). Se sostituiamo *n* con 5, sorgente con *A*, bersaglio con *B* e supporto con *C*, allora la procedura di prima diventa

TORRE (5, A → B*):*
  TORRE (4, *A → C)*
  MOSSA (5, *A → B*)
  TORRE (4, *C → B*)

In altri termini, il programma deve prima riuscire a spostare i primi quattro dischi dal perno *A* al perno *C*. Esso registra il fatto che, quando questa procedura è completata, deve passare a eseguire MOSSA (5, *A → B*), cioè spostare il quinto disco, il più grande, dal perno *A* al perno *B*. Esso registra anche un'altra esecuzione della procedura TORRE, questa volta per spostare i primi quattro dischi dal perno *C* al perno *B*.

A ogni chiamata della procedura TORRE risultano chiamate altre tre procedure: TORRE, poi MOSSA e poi ancora TORRE. La procedura MOSSA non può essere eseguita finché non è stata completata la prima procedura TORRE. Questo significa che in effetti l'ordine che il calcolatore segue è eseguire TORRE per quattro volte successive, operando in questo modo lungo il lato sinistro del diagramma finché incontra

TORRE (1, *A → B*)
MOSSA (2, *A → C*)
TORRE (1, *B → C*)

Un vero programma conterrebbe un'altra istruzione che dice al calcolatore che, quando oggetto di TORRE è un unico disco, esso deve venir mosso senza ulteriori operazioni ricorsive: più precisamente, il primo disco viene mosso dal

perno A al perno B. La procedura MOSSA fa poi sì che il secondo disco venga spostato dal perno A al perno C. Infine il calcolatore sposta di nuovo il primo disco, questa volta dal perno B al perno C, completando la terza mossa. Ora il calcolatore ha completato anche la prima procedura TORRE del riquadro vicino al fondo dell'illustrazione della pagina a fronte. Esegue poi MOSSA $(3, A \rightarrow B)$, naturalmente spostando immediatamente il terzo disco dal perno A al perno B. Successivamente l'istruzione TORRE $(2, C \rightarrow B)$ si sviluppa nelle tre mosse riportate nel riquadro sul fondo della figura.

Le sette mosse così effettuate completano anche l'esecuzione di TORRE al secondo livello del diagramma e il calcolatore continua a seguire lo stesso schema, a volte scendendo a un riquadro a basso livello, a volte ritornando a un livello superiore. Alla fine riesce a farsi strada nell'intero diagramma e il rompicapo è risolto.

La ricorsività sembra spesso magica perché tutta la contabilità necessaria per ricordarsi «dove si è» è tenuta dal calcolatore; gli uomini non sono all'altezza di compiti mnemonici così rilevanti. Per fortuna è disponibile una tecnica che richiede poco impegno della memoria. Quattro anni fa Peter Buneman, dell'Università della Pennsylvania, e Leon Levy, degli AT&T Bell Laboratories, hanno trovato una semplice configurazione di mosse. Buneman e Levy suggeriscono una semplice alternanza tra due tipi di mossa:

1. Spostare il disco più piccolo dal perno che occupa al perno adiacente in senso orario.
2. Spostare un disco qualsiasi tranne il più piccolo.

Il secondo passo non è arbitrario quanto sembra: c'è sempre una sola mossa consentita se si segue questa limitazione - finché di colpo il rompicapo è risolto.

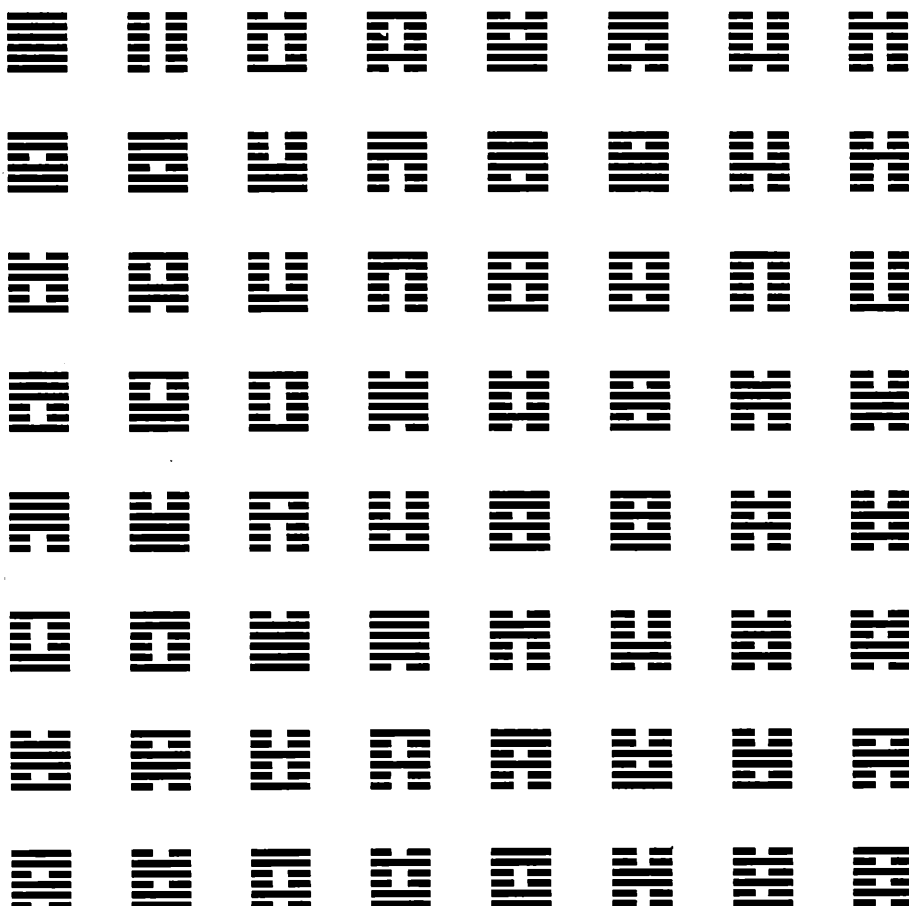
Recentemente ho costruito una torre di otto dischi in legno e ho fatto giocare per un po' un amico. Non concludeva nulla e uscì per un po' dalla stanza. In tutta fretta spiegai la soluzione di Buneman-Levy a sua figlia di otto anni che stava guardando affascinata. Al suo ritorno, il mio amico rimase a bocca aperta nel vedere sua figlia che, con calma e senza esitazione, trasferiva dischi da un piolo all'altro. La bambina completò il rompicapo in pochi minuti. «In gamba, la ragazzina», dissi.

La soluzione di Buneman e Levy si basa sul fatto che in realtà la ricorsività non è necessaria per risolvere il rompicapo della torre; basta una semplice soluzione iterativa. Un programma iterativo è un programma che esegue un compito ripetitivo utilizzando un semplice ciclo invece che una successione di operazioni ricorsive. Anche se sono dotati del particolare fascino della brevità e dell'eleganza, i programmi ricorsivi richiedono grandi capacità di memoria. Per esempio, dalla figura che illustra la soluzione del rompicapo della torre risulta evidente che è necessaria molta memoria per salvare tutte le esecuzioni incomplete di TORRE. Il genere di programma iterativo che si basa sull'algoritmo di Buneman-Levy non richiede quasi per nulla memoria. È raro, però, che si possa sostituire in questo modo un programma ricorsivo con un programma iterativo.

La ricorsività e l'iterazione formano uno dei molti poli del calcolo, una specie di yin e yang nel modo di affrontare il processo ripetitivo. I simboli yin e yang rappresentavano la dualità complementare che è al centro delle principali tradizioni filosofiche della Cina prerivoluzionaria. I due principi formano la base dell'*I Ching* (o *I King*, «Libro dei cambiamenti»). Yin e yang sono come i numeri binari 0 e 1, fondamentali per il calcolo digitale. Nell'*I Ching*, yin è rappresentato da una linea orizzontale spezzata (- -) e yang da una non spezzata (—). Questi due simboli sono combinati a gruppi di sei, che costituiscono 64 esagrammi *(si veda l'illustrazione di questa pagina)*. Se correttamente interpretato, ogni esagramma rappresenta una particolare scelta. Il credente tira delle pagliuzze per stabilire qual è l'esagramma rilevante per la sua vita. La configurazione illustrata è fatta risalire al re Wen, che governò nel 1150 a.C. (La ragione di questo ordinamento degli esagrammi è oscura da tempo. Sarei grato a chi sapesse trovare la chiave di questa misteriosa configurazione.)

Citando i numeri binari, mi è venuto in mente un altro modo per risolvere il rompicapo della Torre di Hanoi. Se si numerano i dischi con 1, 2, 3,... fino a n, sempre dal più piccolo al più grande, si scopre che ogni mossa della soluzione del rompicapo può essere indicata da un numero binario. Per esempio, per risolvere il rompicapo con cinque dischi che ho proposto a titolo di esempio, elencheremmo i numeri binari con cinque bit nel solito ordine di conto. I primi otto numeri binari con cinque bit sono

0 0 0 0 0
0 0 0 0 1 (1)
0 0 0 1 0 (2)
0 0 0 1 1 (1)
0 0 1 0 0 (3)
0 0 1 0 1 (1)
0 0 1 1 0 (2)
0 0 1 1 1 (1)
0 1 0 0 0 (4)

Ogni numero binario con un predecessore nella sequenza ha anche esattamente un bit trasformato da uno 0 in un 1. La posizione di questo bit (contando da destra) è data dal numero decimale scritto a fianco di quello binario. Questi numeri sono anche i numeri delle prime sette mosse dei dischi; la corrispondenza vale per tutta la successione della soluzione standard. Con uno qualsiasi dei numeri binari della successione, Timothy R. S. Walsh, del mio dipartimento all'Università dell'Ontario occidentale, riesce a ricostruire al calcolatore l'esatto aspetto del

*Disposizione dei 64 esagrammi dell'***I Ching** *ideata dal re Wen*

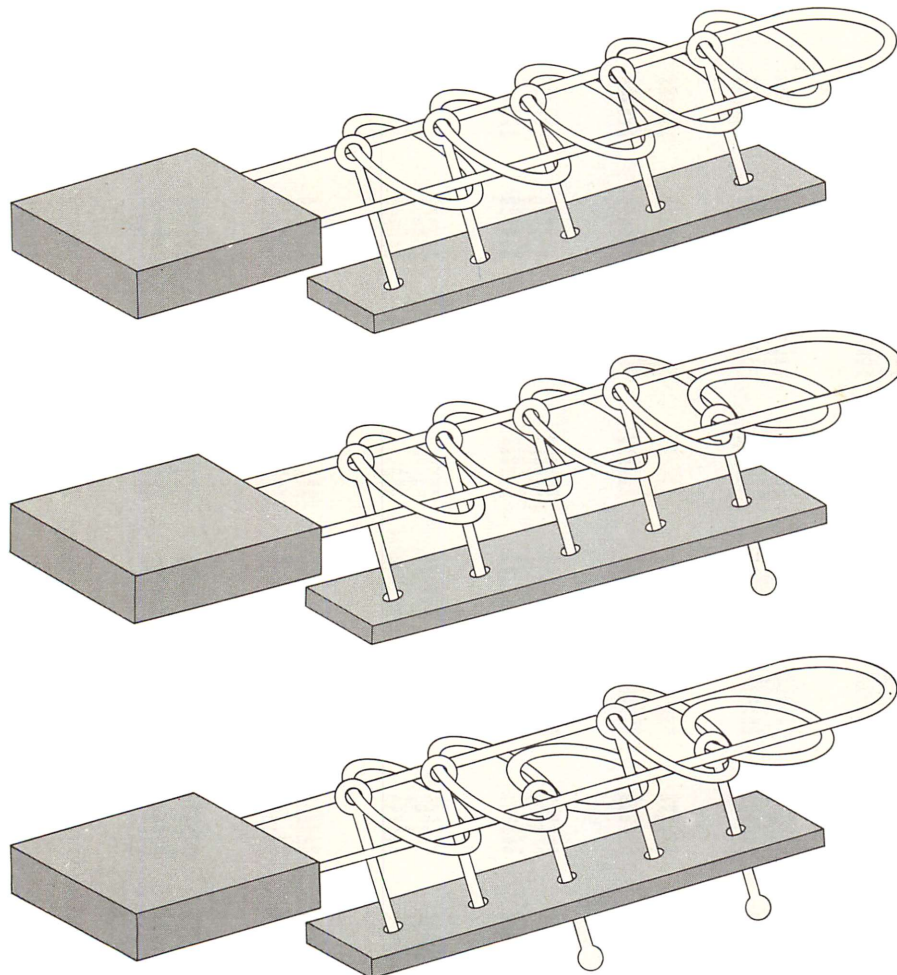
rompicapo della torre a quello stadio. Purtroppo il suo algoritmo è troppo lungo per poterlo dare in questa sede.

L'accenno a yin e yang serve a introdurre gli Anelli cinesi *(si veda la figura di questa pagina)*. Questo rompicapo è formato da un lungo filo metallico a cappio disposto su un'impugnatura con una serie di anelli che circondano il cappio. Ogni anello è collegato da un sostegno metallico a una piattaforma di legno sotto il cappio. Il sostegno che collega ogni anello alla piattaforma passa attraverso l'anello retrostante (più vicino all'impugnatura), impedendone la rimozione dal cappio.

L'obiettivo è togliere tutti gli anelli. Come nel rompicapo della torre, i principianti scopriranno che è molto facile fare mosse sbagliate. La figura mostra le prime due mosse del rompicapo con cinque anelli. Per togliere un anello dal cappio, far scivolare indietro il cappio, se possibile, fino al sostegno dell'anello subito davanti a quello da togliere. Quest'ultimo può allora essere ruotato verso l'alto in modo che la parte ruotante liberi l'estremità del cappio. Facendo di nuovo scivolare in avanti il cappio, l'anello può ora venir piegato lateralmente e fatto scendere attraverso il cappio. Spesso, risolvendo il rompicapo, è necessario rimettere anelli sul cappio; in questo caso si segue il procedimento inverso.

I tentativi di risolvere il rompicapo degli anelli mettono in rilievo lo stesso problema generale posto dal rompicapo della torre: compaiono configurazioni in cui sono stati tolti dal cappio un certo numero di anelli consecutivi, il che porta a ritenere che ci sia qualche modo per togliere tutti gli anelli. Non sorprende, quindi, apprendere che il rompicapo degli anelli può essere risolto con lo stesso genere di algoritmo. In realtà, c'è anche un semplice procedimento iterativo per risolvere il rompicapo degli anelli, procedimento notevolmente più limpido di quello per risolvere il rompicapo della torre. Sarebbe un peccato dare qui quella semplice tecnica e negare ai lettori la loro esperienza «aha!». Non sono in grado di dare alcun suggerimento senza fornire direttamente la soluzione, tuttavia posso dire che la soluzione può essere formulata in due o tre fasi e che non richiede alcuna particolare notazione.

Un po' più sorprendente, invece, è la quasi identità dei due rompicapo, identità che ci richiama un fenomeno abbastanza comune nella teoria della computazione e in matematica: due problemi che a un esame superficiale sembrano essere notevolmente diversi si rivelano a un esame più ravvicinato, sostanzialmente uguali!

Il collegamento tra i due rompicapo è dato da due codici binari e da un algoritmo che traduce uno nell'altro. Proprio come rappresentiamo le mosse del rompicapo della torre con numeri binari consecutivi, così introduciamo un nuovo codice binario per rappresentare il rompicapo degli anelli: un 1 rappresenta un anello sul cappio e uno 0 simbolizza un anello liberato dal cappio. Il rompicapo con cinque anelli può allora essere rappresentato da una successione a cinque cifre fatta di 0 e 1, dove la cifra più a sinistra sta per l'anello subito dopo l'impugnatura. Scritte come successione di numeri di questo genere, le prime quattro configurazioni del rompicapo degli anelli sono le seguenti:

1 1 1 1 1 (tutti gli anelli su)
1 1 1 1 0 (primo anello giù)
1 1 0 1 0 (primo e terzo anello giù)
1 1 0 1 1 (terzo anello giù)

Nelle due mosse successive vengono tolti il secondo anello e poi il primo. Subito dopo viene tolto il quinto anello. Si rimettono poi i primi tre anelli sul cappio per prepararsi a togliere il quarto anello. In tutto, bastano 21 mosse per togliere tutti i cinque anelli. Questo è l'elenco delle ultime quattro configurazioni:

0 0 0 1 0 (secondo anello su)
0 0 0 1 1 (primo e secondo anello su)
0 0 0 0 1 (primo anello su)
0 0 0 0 0 (nessun anello su)

I lettori possono divertirsi a trovare i 14 numeri di codice mancanti e a risolvere il rompicapo in questa forma, seguendo due semplici regole che rappresentano il vincolo che due anelli adiacenti pongono l'uno all'altro:

1. La cifra più a destra si può modificare in ogni momento (da 0 a 1 o da 1 a 0).
2. L'unica altra cifra che si può modificare è quella immediatamente a sinistra dell'1 più a destra.

A prima vista non sembra sussistere alcuna relazione ovvia tra la successione di 21 numeri di codice binari a cui si è appena fatto riferimento e la successione binaria che nasce dal rompicapo della torre. In realtà la relazione esiste e chiama in causa il codice di Gray, così chiamato dal nome dell'ingegnere Frank Gray che lo inventò negli anni trenta, quando lavorava agli AT&T Bell Laboratories, per fornire una tecnica di correzione degli errori per le comunicazioni elettroniche. Il codice è illustrato nella figura della pagina a fronte. Nella stessa figura si vedono i primi 22 numeri binari e, a fianco, i corrispettivi nel codice di Gray.

L'esame dei numeri del codice di Gray in figura rivela una coincidenza notevole. In ordine inverso, i numeri del codice di Gray sono proprio le posizioni successive degli anelli in una soluzione del rompicapo degli anelli cinesi!

Queste, allora, sono le 21 posizioni che formano una soluzione al rompicapo dei cinque anelli. Ma il rompicapo della torre con cinque dischi richiede 31 mosse per essere completato. Si potrebbe pensare

Le prime due mosse del rompicapo degli Anelli cinesi

che, se i due rompicapo sono in un certo senso identici, le loro soluzioni dovrebbero avere lo stesso numero di mosse. La discrepanza, però, scompare se si considerano i numeri del codice di Gray al di là del 21esimo. Ognuno di essi rappresenta una possibile configurazione del rompicapo degli anelli e l'ultimo (corrispondente al numero binario 11111) è 10000, la configurazione in cui solo l'ultimo anello è sul cappio. Questo implica che, se si vuole far penare qualcuno sul rompicapo degli anelli, bisogna presentarglielo con tutti gli anelli tolti tranne l'ultimo. In questo caso, il numero di mosse per risolvere il rompicapo con n anelli è 2^{n-1}, esattamente come nel rompicapo con n dischi.

Recentemente Sydney N. Afriat, professore di economia e matematica all'Università di Ottawa, ha scritto un libro ricco di informazioni sugli anelli cinesi. Il titolo è *The Ring of Linked Rings* ed è pubblicato da Gerald Duckworth & Co. Ltd., The Old Piano Factory, 43 Gloucester Crescent, Londra NW1, Inghilterra. Devo ad Afriat l'idea di collegare il rompicapo degli anelli cinesi alla dualità yin e yang. Pur sospettando che abbiano origini cinesi, Afriat è riuscito a trovare tracce precise degli Anelli cinesi solo a partire dal 1550. Il suo libro descrive anche il «Codice Gros», un precursore del codice Gray che risale al XIX secolo ed è dovuto al matematico francese Louis A. Gros, che pubblicò un trattato sul rompicapo nel 1872. Detto per inciso, i francesi chiamano questo rompicapo «Le Baguenodier» e gli inglesi «The Tiring Irons».

Entrambi i rompicapo presentano numerosi altri aspetti che, per ragioni di spazio, non posso prendere in considerazione in questa sede.

A titolo di esempio, Leroy J. Diskey, del Dipartimento di matematica pura dell'Università di Waterloo, nell'Ontario, mi ha fatto presente che risolvere il rompicapo degli anelli cinesi è equivalente anche a far slittare gli spigoli di un ipercubo a n dimensioni.

Si potrebbe peraltro osservare che il rompicapo della torre con n dischi può essere risolto in 2^{n-1} mosse se si utilizzano i tre perni, mentre l'impiego di $n + 1$ perni abbrevia decisamente il numero complessivo di mosse necessarie portandolo a un numero notevolmente più piccolo, $2n - 1$.

Che cosa avviene passando da tre a $n + 1$ perni?

Come cambia, inoltre, il numero minimo di mosse da eseguire nell'ambito di una soluzione al variare del numero di perni disponibili?

	CODICE BINARIO	CODICE DI GRAY		CODICE BINARIO	CODICE DI GRAY
0	0 0 0 0 0	0 0 0 0 0	11	0 1 0 1 1	0 1 1 1 0
1	0 0 0 0 1	0 0 0 0 1	12	0 1 1 0 0	0 1 0 1 0
2	0 0 0 1 0	0 0 0 1 1	13	0 1 1 0 1	0 1 0 1 1
3	0 0 0 1 1	0 0 0 1 0	14	0 1 1 1 0	0 1 0 0 1
4	0 0 1 0 0	0 0 1 1 0	15	0 1 1 1 1	0 1 0 0 0
5	0 0 1 0 1	0 0 1 1 1	16	1 0 0 0 0	1 1 0 0 0
6	0 0 1 1 0	0 0 1 0 1	17	1 0 0 0 1	1 1 0 0 1
7	0 0 1 1 1	0 0 1 0 0	18	1 0 0 1 0	1 1 0 1 1
8	0 1 0 0 0	0 1 1 0 0	19	1 0 0 1 1	1 1 0 1 0
9	0 1 0 0 1	0 1 1 0 1	20	1 0 1 0 0	1 1 1 1 0
10	0 1 0 1 0	0 1 1 1 1	21	1 0 1 0 1	1 1 1 1 1

Ogni numero del codice di Gray formato da cinque cifre è ottenuto dal corrispondente numero binario attraverso l'applicazione di una semplice regola: se si numerano le cifre da sinistra a destra, la prima cifra nel codice di Gray è sempre uguale alla prima cifra binaria. Poi ogni cifra di Gray è un 1 se la corrispondente cifra binaria è diversa da quella successiva; altrimenti è uno 0.

Codice di Gray per i primi 22 numeri binari

Soluzioni proposte

Per risolvere il rompicapo degli anelli cinesi si alterna il movimento del primo anello (mettendolo o togliendolo) con quello di qualche altro anello. Ogni volta, solo un altro anello può essere mosso. Se il numero degli anelli è dispari, si parte con il primo anello; in caso contrario con l'altro. Così Rob Hardy di Dayton, Ohio, riassume questa semplice soluzione:

Una soluzione iterativa
Non è più certo un divertimento:
Alternate il cambio dell'ultimo anello
Mettendone un altro in movimento.

Nessuno ha trovato un semplice schema matematico dietro l'ordinamento del re Wen per gli esagrammi dell'*I Ching*. Molti hanno osservato che gli esagrammi sono accoppiati attraverso le colonne in modo semplice. Homer E. Brown, ingegnere elettromeccanico di Cary, North Carolina, ha prodotto un'analisi suggestiva: a partire da ogni esagramma contare verso l'alto o verso il basso 10 esagrammi, passando dalla fine di una colonna all'inizio della successiva, se necessario. L'esagramma che si ottiene ha sempre una relazione semplice con quello di partenza, ma la regola non è deterministica perché non si sa mai se contare verso l'alto o il basso.

A dispetto di questa apparente relazione (che ha molte eccezioni), sono incline a ritenere che la soluzione del re Wen segua principi metafisici. Il tentativo di proiettare le attuali preoccupazioni riguardo la scienza e la tecnologia sulle culture del passato dà luogo a una visione distorta. Come regola generale, i sistemi «scientifici» di una volta svolgevano un ruolo strettamente ancillare rispetto a una visione del mondo fortemente religiosa. Opinioni analoghe sono proposte da Bernard X. Bovasso di Saugerties, New York. Lo *I Ching* si preoccupa dell'ordine del tempo, sostiene Bovasso, e così era per il re Wen. David White, un filosofo del Macalester College di St. Paul, Minnesota, ha fatto riferimento a una traduzione inglese dell'*I Ching*, opera di James Legge. In un'appendice di quest'opera si traccia una visione metafisica che governerebbe l'ordine di tutti i 64 esagrammi. Per esempio, i primi tre sono collegati rispettivamente ai concetti di cielo, terra e caos. L'esagramma del caos, che denota ciò che viene considerato il disordine di tutte le cose create prese collettivamente, segue il cielo e la terra dato che tutte le cose create riempiono lo spazio tra cielo e terra.

Il re (un programma per gli scacchi) è morto. Viva il re (una macchina per gli scacchi)!

di A. K. Dewdney

Le Scienze, aprile 1986

Se CRAY BLITZ fosse in grado di ricordare qualcos'altro oltre le mosse degli scacchi, non dimenticherebbe mai la sera del 15 ottobre 1985. In quella sera si è tenuta l'ultima tornata del Campionato nordamericano di scacchi per calcolatori, in occasione del convegno annuale dell'Association for Computing Machinery. Cinque tavoli, separati dal pubblico da una barriera, sono sistemati all'inizio di una sala del Radisson Hotel di Denver. A ciascun tavolo si fronteggiano due squadre di programmatori ed esperti che un po' giocano, un po' rimangono in posizione di attesa.

Il torneo prevede la partecipazione di 10 contendenti al titolo, dai nomi alquanto bizzarri: AWIT, BEBE, CHAOS, CRAY BLITZ, HITECH, INTELLIGENT SOFTWARE, LACHEX, OSTRICH, PHOENIX e SPOC (*si veda l'illustrazione della pagina a fronte*). Sono assenti tre grandi nomi che hanno dominato gli scacchi al calcolatore negli anni scorsi: BELLE, CHESS 4.7 e NUCHESS.

L'interesse è incentrato sulla partita finale tra CRAY BLITZ e HITECH. Al tavolo, dalla parte di CRAY BLITZ, stanno Robert Hyatt dell'Università del Southern Mississippi, Albert Gower, un esperto di scacchi dello stesso ateneo, e Harry Nelson del Lawrence Livermore National Laboratory. I loro avversari sono Hans Berliner della Carnegie Mellon University e Murray Campbell, uno dei suoi studenti che è un giocatore esperto. Nella squadra di HITECH, Berliner ricopre il duplice ruolo di programmatore e di esperto. Col procedere della gara e il crescere della tensione, Berliner si alza spesso, con un sorriso stanco sul volto. Una volta passa vicino alla mia sedia e mi sussurra: «È proprio come ai tempi dei miei campionati nazionali!» (Per parecchi anni, tra la fine degli anni cinquanta e l'inizio degli anni sessanta, Berliner si è qualificato tra i primi dodici giocatori degli Stati Uniti.)

A differenza di quel che succede ai campionati degli Stati Uniti, dove regna un silenzio mortale, in questo torneo si conversa, talvolta si ride, si sente il rumore continuo delle tastiere e l'incessante commento dell'arbitro Michael Valvo, un eccentrico consulente di informatica e maestro internazionale di scacchi di Sedona in Arizona. «Mossa debole da parte del nero. Il re è ancora troppo esposto e i due pedoni in c5 e c6 continuano a ostacolare la difesa.» Lì accanto un membro della squadra di CRAY BLITZ esclama, senza rivolgersi a qualcuno in particolare: «Divertente! Pensavo che avrebbe messo il re in f3.»

Durante tutta questa tornata finale è risultato chiaro che HITECH ha il sopravvento sul suo rivale: ben presto CRAY BLITZ è caduto in un *zugzwang*, una posizione critica da cui si può uscire solo con una brutta mossa o con una perdita di pezzi. In questo caso, CRAY BLITZ è costretto a disporre male i suoi pedoni e HITECH continua a sfruttare il vantaggio.

A mezzanotte non è ancora finita. La maggior parte delle partite sono terminate e gli esperti affermano che HITECH ha vinto. La squadra di CRAY BLITZ chiede al giudice Valvo il permesso di abbandonare e Valvo propone di fare altre due mosse: se la posizione di CRAY BLITZ non migliora, la squadra può abbandonare. Così avviene: HITECH è campione nordamericano e di fatto re degli scacchi al calcolatore.

Si ride e si continua a conversare. L'assenza di BELLE, CHESS 4.7 e NUCHESS è significativa? Dice un organizzatore del torneo: «Sarebbe stato interessante se BELLE e qualche altro programma avessero partecipato, ma non credo che il risultato sarebbe stato molto diverso». Prosegue precisando che, in termini di programmi e di macchine, sostanzialmente non c'è differenza tra campionato nordamericano e campionato del mondo. Il discorso si sposta su Kasparov e Karpov e poi sulla teoria. Dice un partecipante, evidentemente degno di considerazione: «Non sto scherzando. Un programma a 20 livelli che si preoccupi solo dei pezzi può battere qualsiasi gran maestro». L'affermazione suscita un po' di discussione, ma dopo qualche minuto la sala si svuota: il campionato nordamericano è terminato.

La battuta sul programma a 20 livelli è interessante. Il gioco degli scacchi si può rappresentare come un grande albero fatto di linee e nodi. Lo rappresento rovesciato cosicché il nodo radice si trovi in alto. Ciascun nodo rappresenta una posizione possibile, vale a dire una scacchiera sulla quale i pezzi e i pedoni siano arrivati alle loro posizioni attraverso mosse legittime. Un nodo è congiunto da una linea a un nodo discendente, se lo spostamento di un singolo pezzo o di un pedone trasforma la scacchiera rappresentata dal primo nodo in quella rappresentata dal secondo. Una partita di scacchi si può sempre identificare con un particolare cammino lungo l'albero degli scacchi, a partire dal nodo radice (quando non è stata fatta alcuna mossa) giù lungo l'albero fino a un nodo dove, come regola generale, sono rimasti pochi pezzi e uno dei giocatori ha subito scacco matto o è stato costretto ad abbandonare.

Un programma che gioca a scacchi cerca di esplorare, dell'albero, solo quel tanto che serve. Dal nodo che rappresenta la posizione attuale esamina tutte le scacchiere discendenti (livello 1), poi le discendenti delle discendenti (livello 2) e così via. La profondità media della sua esplorazione viene detta «previsione». In questa misura è compresa la maggior parte di quella che si potrebbe chiamare l'intelligenza del programma. In parte minore, essa consiste nella valutazione delle scacchiere che costituiscono l'orizzonte della sua previsione. Il programma le analizza e attribuisce a ciascuna un valore numerico che ne riflette l'appetibilità. Con una procedura chiamata minimax, il programma fa risalire alcuni valori lungo l'albero fino ai nodi del livello 1. Il nodo che riceve il valore più alto indica il gioco da svolgere.

Vi è un'interessante complementarità tra le due parti dell'intelligenza del programma: tanto migliore è lo schema di valutazione, tanto meno profonda deve essere l'esplorazione dell'albero. In effetti, se il programma avesse uno schema di valutazione perfetto, non avrebbe bisogno di esplorare più a fondo del primo livello. Al contrario, un programma con uno schema di valutazione molto semplice deve indagare molto più in profondità, se vuole giocare in maniera efficace. A quale profondità deve scendere una ricerca che si preoccupi solo dei pezzi, per essere efficace contro un gran maestro? È sufficiente una ricerca di venti livelli?

Il titolo di gran maestro è conferito dalla Fédération Internationale des Echecs a giocatori che si sono distinti in campo internazionale. (La federazione non prende in considerazione i calcolatori.) Un gran maestro ha generalmente punteggi superiori a 2400, il livello di un maestro anziano. Fino al Campionato nordamericano di scacchi per calcolatori, HITECH aveva giocato 21 partite in tornei umani, guadagnando 2233 punti. Era, cioè, il più quotato fra i calcolatori che giocano a scacchi in tutto il mondo. Secondo Berliner, che ai tempi in cui gareggiava aveva un punteggio di 2443, la quotazione di HITECH è cresciuta con una media di otto punti a partita nei tornei internazionali. Si può supporre che, dopo altre 14 partite, la macchina sorpasserà il suo ideatore?

Tutto questo porta a porsi la domanda:

quanto bravi possono diventare i calcolatori che giocano a scacchi? Un calcolatore diventerà mai il migliore giocatore del mondo? David Levy, un tempo giocatore e attualmente autore e imprenditore, ha affidato la questione a una serie di scommesse. Nel 1968 scommise 500 sterline con John McCarthy della Stanford University che nessun calcolatore lo avrebbe battuto nei successivi 10 anni. Levy incassò la posta nell'agosto del 1978 in occasione della Canadian National Exhibition di Toronto, dove giocò con CHESS 4.5, un programma elaborato alla Northwestern University. La scommessa iniziale venne, poi, rinnovata per un periodo di altri sei anni e per 6000 dollari. Nell'aprile 1984 Levy da Londra ha giocato per telefono con CRAY BLITZ e ha vinto di nuovo.

Tutto questo lo ha spinto a scommettere, a Denver, 100 000 sterline che nei prossimi 10 anni qualunque calcolatore che raccolga la sfida verrà battuto da un giocatore umano scelto da lui. Se qualcuno raccoglierà la sfida di Levy, sarà probabilmente non un semplice programma, ma un calcolatore specializzato. Finora non si è fatto avanti alcuno sfidante.

I due finalisti del torneo nordamericano, HITECH e BEBE, erano proprio macchine di questo tipo. Curiosamente, il concorrente dello stesso Levy, un programma chiamato INTELLIGENT SOFTWARE, si è classificato terzo. Gira su un Apple IIe che non ha nulla di più sofisticato di una scheda acceleratrice, che ne raddoppia la velocità. Forse Levy ha sviluppato uno schema di valutazione superiore.

Gli esperti di scacchi presenti al campionato sono d'accordo nel ritenere che la miglior partita del torneo sia stata quella della seconda tornata tra CRAY BLITZ e BEBE, un prodotto dell'impresa privata. Tony Scherzer, la cui azienda, SYS-10 Inc., ha sviluppato BEBE, negli ultimi anni ha portato la sua creazione a parecchi tornei. BEBE non è un semplice programma, ma una vera e propria macchina che gioca a scacchi. La partita è stata significativa non solo perché era la più interessante del torneo, ma anche perché era la prima partita persa da CRAY BLITZ in tre anni.

I lettori con una scacchiera possono seguire la partita di CRAY BLITZ contro BEBE giocando le 50 mosse elencate nel seguito. I pezzi sono indicati da lettere maiuscole: R, re; D, regina; A, alfiere; C, cavallo; T, torre. Le caselle della scacchiera sono contrassegnate dalle coordinate lettere-numeri. Quando la scacchiera è nella posizione standard, cioè con la casella in basso a sinistra nera, le colonne sono contrassegnate da sinistra a destra con le lettere da *a* fino a *h*; le traverse sono numerate da 1 a 8 a cominciare dal basso della scacchiera. La notazione impiegata nella presentazione delle partite può andare dalla semplice formula Rb1 (re nella casella b1) alla problematica Cf3 (cavallo in f3). Quale cavallo? In quella particolare mossa solo un cavallo può spostarsi in f3. La mossa di un pedone viene indicata designando una casella, per esempio e4.

La partita è commentata da Valvo.

	CRAY BLITZ (Bianco)	BEBE (Nero)
1.	e4	c5
2.	Cf3	d6
3.	d4	cxd4

(La x significa che un pezzo o un pedone viene mangiato.)

4.	Cxd4	Cf6
5.	Cc3	g6
6.	Ag5	Ag7
7.	Dd2	Cc6
8.	0-0-0	0-0

(Arrocco lungo del bianco, arrocco corto del nero.)

9.	Cb3	Te8
10.	Ac4	Cg4

Il nero ha giocato Cg4 pensando di fare Ac3xC alla mossa successiva. Il nero può aver pensato il bianco obbligato a 12 c3xA, ma il nero cambia parere alla successiva mossa del bianco. Se Axc3, allora Dxc3!; Cxf2 non funziona se l'una o l'altra delle due torri viene portata in f1, mentre Axf7, scacco, sarebbe fatale.

11.	h3	Cge5
12.	Ab5	a6
13.	Ae2	a5
14.	Ab5	Ae6
15.	Cd5	a4
16.	Cd4	Ad7

La situazione del bianco è disperata. Il pedone nero in a4 minaccia di creare una difficile situazione di debolezza intorno al re bianco.

17.	Cxc6	bxc6
18.	Cxe7 scacco	Txe7
19.	Axe7	Dxe7
20.	Ae2	De6
21.	Rb1	Tb8

(Nell'illustrazione di pagina 24 si può vedere la situazione della scacchiera a questo punto.) Il nero minaccia 22...Tb2 scacco, seguito da 23 Rb2 Cc4, una forchetta che elimina la regina bianca.

22.	b3	axb3
23.	cxb3	Ae8
24.	Rc2	Cd7
25.	f3	Ta8
26.	Rc1	Cc5
27.	Dc2	Df6
28.	Ac4	Da1 scacco
29.	Rd2	Dxa2

Una mossa ancora più forte è 29...Ac3, scacco, seguita da 30 Re2 Ta2!

30.	Dxa2	Txa2 scacco
31.	Rc1	d5
32.	exd5	cxd5
33.	Axd5	Ab5
34.	The1	Cd3 scacco

PROGRAMMA	ORIGINE	CALCOLATORE	LINGUAGGIO	POSIZIONI AL SECONDO	LIVELLI DI PREVISIONE
AWIT	Università dell'Alberta	Amdahl 5860	Algol W	10	3
BEBE (secondo)	SYS-10, Inc., Hoffman Estates, Illinois	Macchina personalizzata	Assembler	20 000	7
CHAOS	Università del Michigan	Amdahl 5860	FORTRAN	70	4
CRAY BLITZ	Università del Southern Mississippi	Cray X-MP 48	FORTRAN/ Assembler	100 000	8
HITECH (primo)	Carnegie-Mellon University	Sun con circuiti VLSI personalizzati	C	175 000	8
INTELLIGENT SOFTWARE (terzo)	Intelligent Soft-Ware Inc., Londra	Apple II e con scheda acceleratrice	Assembler	500	7
LACHEX	Los Alamos National Laboratory	Cray X-MP 48	FORTRAN/ Assembler	50 000	7
OSTRICH	McGill University	Rete di sette Nova e un Eclipse	Assembler	1200	6
PHOENIX	Università dell'Alberta	Rete di VAX 780 e 10 Sun	C	540	6
SPOC	SDI/Cypress Software, San Jose, California	IBM PC	Assembler	300	5

I partecipanti al Campionato nordamericano di scacchi per calcolatore

Il vantaggio (un pezzo) del nero sta per essere aumentato da un altro scambio. In un torneo tra uomini a questo punto sarebbe ragionevole per il bianco abbandonare.

35.	Txd3	Axd3
36.	Te8 scacco	Af8
37.	g4	Rg7
38.	Te3	Aa3 scacco
39.	Rd1	Ta1 scacco
40.	Rd2	Af1
41.	Rc3	Tc1 scacco
42.	Rd2	Tc5
43.	Re1	Axh3
44.	Ac4	h5
45.	gxh5	gxh5
46.	Rf2	h4
47.	Td3	Af5
48.	Td4	h3
49.	Th4	Tc7
50.	Th5 (chiede di abbandonare)	

Il programma CRAY BLITZ gira su un calcolatore Cray XM-P 48. Famoso per la sua velocità come multielaboratore, il Cray è nondimeno un calcolatore di uso generale e non una macchina per gli scacchi. BEBE, i cui circuiti sono dedicati agli scacchi, ha ovviamente superato, nella partita riportata qui sopra, la combinazione Cray-CRAY BLITZ.

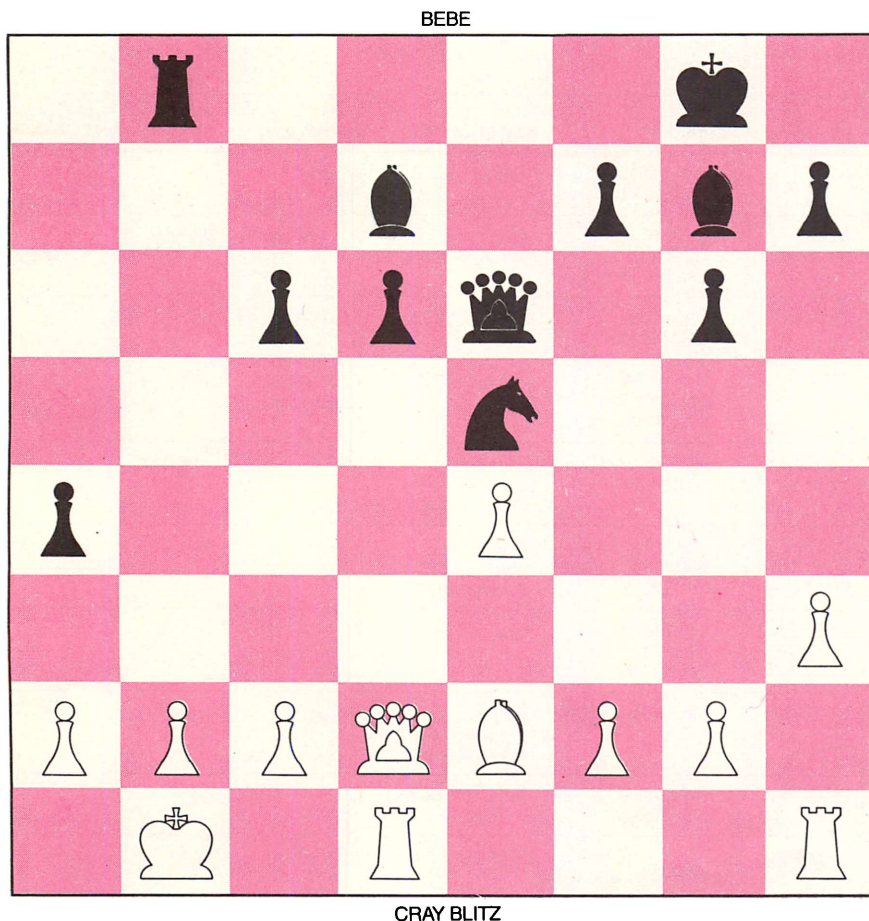
HITECH in un certo senso è più specializzato. Quando la Carnegie-Mellon University era il Carnegie Institute of Technology, vi venne messo a punto un programma che giocava a scacchi, chiamato TECH. Il nome HITECH è legato al fatto che Berliner, Campbell e gli altri membri del gruppo HITECH (Carl Ebeling, Gordon Goetsch, Andy Palay e Larry Slomer) hanno fatto rivivere la tradizione di TECH in un mondo di integrazione a grandissima scala (VLSI: *very large scale integration*) e di crescente parallelismo. La macchina HITECH comprende un calcolatore Sun dotato di un elaboratore appositamente progettato, chiamato il ricercatore. Sul calcolatore Sun girano tre programmi: un'interfaccia utente, un controllore di compito e un oracolo. L'oracolo comprende quello che gli esperti di scacchi con il calcolatore chiamano il libro, ossia un grande catalogo di aperture e variazioni che fanno parte del bagaglio comune dei giocatori esperti di scacchi. La base di dati dell'oracolo contiene molte altre nozioni scacchistiche, che si possono facilmente espandere. Quando il ricercatore esamina le possibilità di gioco a partire da una posizione data, procede sulla base delle informazioni rilevanti rispetto a quella posizione ricavate dall'oracolo.

Il ricercatore stesso contiene un microelaboratore e parecchi moduli hardware che generano mosse, le valutano, controllano le mosse ripetute e così via. Il microelaboratore coordina la loro attività. Il generatore di mosse è fatto di 64 chip VLSI, uno per ciascuna casella della scacchiera. Ciascun chip esamina l'intera scacchiera per stabilire se un pezzo o un pedone possono essere portati sulla casella sotto il suo controllo. Esso determina la mossa migliore secondo criteri standard come quelli che stabiliscono quando è opportuno catturare un pezzo o controllare il centro. Nello stesso momento gli altri 63 chip stanno facendo la stessa cosa. Questa struttura comporta che, se ci sono 10 pezzi sulla scacchiera, le mosse possibili vengono generate dieci volte più velocemente, mantenendo invariati gli altri fattori.

La valutazione delle mosse deve procedere di pari passo con il processo di generazione delle mosse stesse. Una prima fase di valutazione viene compiuta dal generatore stesso, che ospita una specie di supervisore, il quale giudica le mosse generate dai 64 chip. Ciascun chip calcola un numero che dà una valutazione della potenza della sua mossa migliore e trasmette il numero al supervisore. I numeri generati dai chip sono come urla che chiedono attenzione. Il supervisore li dispone in ordine di volume (leggi: di efficacia).

HITECH procede, quindi, ad analizzare l'albero di gioco secondo l'ordinamento prodotto dal supervisore delle mosse che sono possibili a partire dalla posizione di quel momento. Una seconda fase di valutazione è condotta dal modulo di valutazione in ciascuna nuova posizione che si genera all'interno dell'albero di gioco. Servendosi delle nozioni rilevanti rispetto alla posizione attuale, fornite dall'oracolo, il modulo valuta ciascuna disposizione sulla scacchiera, sia che si trovi, sia che non si trovi nell'orizzonte di previsione. Il parallelismo consiste in questo. Un maggior impegno non comporta un maggior tempo. Il controllore di compito del Sun dice al ricercatore quanto andare a fondo nell'esplorare l'albero e, quando la ricerca è terminata, se procedere ancora oltre. In questo modo HITECH gestisce una previsione media di otto livelli, ma può, se se ne presenta l'occasione, esplorare anche 14 livelli. Questo può sembrare ancora lontano dai 20 livelli necessari per battere un gran maestro. L'uso fatto da HITECH del parallelismo, però, e il raffinato impiego di nozioni scacchistiche nell'esplorazione dell'albero compensano forse la relativamente scarsa profondità dell'indagine. In ogni caso, in occasione di prossimi campionati del mondo di scacchi per calcolatori, HITECH è destinato a diventare inarrestabile.

HITECH è la più recente macchina per scacchi del mondo.

La prima fu inventata nel 1890 da Leonardo Torres y Quevedo, un ingegnere spagnolo. Servendosi di leve meccaniche, pulegge e interruttori elettromeccanici, giocò una mediocre partita di torre e re contro re. Agli uomini venne concesso il privilegio di curare gli affari di un re solo, cercando di evitare di subire scacco matto da parte della potente com-

La scacchiera dopo la ventunesima mossa

binazione di torre e re della macchina. La macchina di Torres y Quevedo non fu mai sconfitta.

I lettori dovrebbero creare una strategia che produca questo risultato. Si deve supporre che il re del giocatore uomo non inizi in una posizione di stallo. Si tratta, quindi, di precisare nel minor numero di regole possibile come fa la macchina a dare scacco matto a partire da una posizione arbitraria. La posizione illustrata nella figura di questa pagina può costituire un punto di partenza.

La macchina è il bianco e muove per prima. Come fanno il re bianco e la torre a costringere il re nero allo scacco matto? Il bianco potrebbe cominciare muovendo la sua torre nella traversa *d*. Questo impedirebbe al nero di spostare il suo re a sinistra. La manovra può essere ripetuta, se il re nero si sposta verso destra, ma che cosa accade se continua a occupare la traversa *e*, andando semplicemente avanti e indietro?

Soluzioni proposte

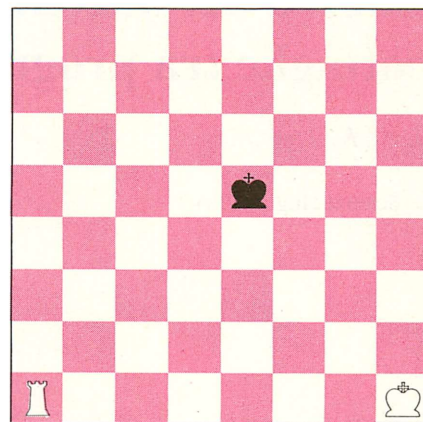
I modi di affrontare il problema, per non parlare degli stili algoritmici, sono stati così differenti che non ho potuto individuare un'unica soluzione come la più rapida. Tre lettori, Scott K. Liddell di Silver Spring, Maryland, Paul Canniff di Marlton, New Jersey, e Stephen J. Perris di Baton Rouge, Louisiana, non solo hanno usato algoritmi simili per arrivare allo scacco matto, ma hanno anche scritto programmi per confermarne l'efficacia.

L'algoritmo di Liddell, il migliore dei tre, applica una strategia di rotazioni e divisioni: «Ruota la scacchiera per trovare ogni configurazione in cui il re nero sia almeno due traverse avanti al re bianco. C'è sempre almeno una configurazione del genere, ma per lo più ce ne sono due. Se l'algoritmo trova una sola di queste configurazioni, la scelta è obbligata. Se vengono trovate due configurazioni, il programma sceglie quella in cui il re nero è più vicino all'estremità della scacchiera. Una volta scelta una posizione, il programma taglia la scacchiera in modo che il re isolato non possa avanzare. Se si trova in una posizione che le impedisce di tagliare l'avanzata del re nero, la torre bianca si sposta all'estremità della scacchiera. Qualsiasi cosa faccia il Nero, il programma è poi in grado di tagliare la scacchiera alla mossa successiva. L'intera procedura richiede almeno due mosse da parte del Bianco.»

Con «intera procedura» ci si riferisce a un ciclo base di mosse con le quali il Nero viene obbligato a spostarsi progressivamente verso un angolo della scacchiera. Un grande vantaggio dell'algoritmo sottostante è l'uso della simmetria di rotazione; se il re nero non si trova «al di sopra» del re bianco, il programma non fa che ruotare la scacchiera fino a ottenere la posizione desiderata.

Numerose fra le strategie proposte dai lettori applicano un concetto chiave degli scacchi: l'opposizione. Se i due re distano solo due caselle uno dall'altro, il re che deve muoversi non si può avvicinare all'altro re: deve o mantenere la stessa distanza oppure allontanarsi. Forse la migliore delle strategie che impiegano una continua opposizione è quella proposta da Alexis A. Gilliland di Arlington, Virginia. Ecco le regole (lievemente parafrasate) che adotta.

1. La torre occupa sempre la prima o l'ottava traversa. Quando viene attaccata su una traversa, si sposta sull'altra.

2. Il re bianco si sposta verso il centro della scacchiera e cerca di spingere il re nero verso la colonna della torre.

Come dare scacco matto con torre e re?

3. Il Bianco cerca una posizione di opposizione in cui i due re si trovino distanti una colonna sulla stessa traversa e tocchi al Bianco muovere. Per arrivare all'opposizione può essere necessario che il Bianco effettui una mossa inutile con la torre, violando temporaneamente la regola 1.

4. Una volta ottenuta l'opposizione, il Bianco dà scacco al re nero con la torre spostandola sulla colonna del re bianco e obbligando così il re nero ad avvicinarsi di una colonna al margine della scacchiera.

5. Si ripetono i passi 3 e 4 finché il re nero arriva al margine della scacchiera. A quel punto il passo 4 porta allo scacco matto.

Gilliland sostiene che la sua strategia conduce inevitabilmente allo scacco matto e afferma anche che difficilmente un algoritmo per arrivare allo scacco matto nel minor numero di mosse ha a sua volta una lunghezza minima.

Guerra dei nuclei: battaglie tra programmi a colpi di bit

di A. K. Dewdney

Le Scienze, luglio 1984

Due programmi per calcolatore nel loro habitat naturale - i chip di memoria di un calcolatore digitale - si inseguono l'un l'altro da indirizzo a indirizzo. Talvolta vanno a cercare il nemico; talvolta dispongono uno sbarramento di bombe numeriche; talvolta trascrivono una propria copia per sfuggire al pericolo o si fermano per riparare i danni. Questo è il gioco che chiamo «Core War», Guerra dei nuclei. A differenza di quasi tutti gli altri giochi con il calcolatore, qui gli uomini non giocano affatto. I programmi che si scontrano sono ovviamente scritti da qualcuno, ma, una volta iniziata la battaglia, il creatore di un programma non può far altro che guardare impotente sullo schermo se il prodotto di ore e ore di progettazione e programmazione vive o muore. Il risultato dipende interamente da quale programma viene colpito per primo in un'area vulnerabile.

Il termine «Guerra dei nuclei» deriva da una tecnologia di memoria ormai superata. Negli anni cinquanta e sessanta il sistema di memoria di un calcolatore era costituito da migliaia di nuclei o anelli ferromagnetici legati su una rete di sottili conduttori. Ciascun nucleo poteva assumere il valore di un bit (una cifra binaria), l'unità minima di informazione. Oggi gli elementi di memoria sono fabbricati su chip di semiconduttore, ma spesso nella lingua inglese la parte attiva del sistema di memoria, quella in cui viene conservato il programma mentre viene eseguito, viene chiamata «*core memory*» (o più semplicemente «*core*», nucleo).

I piani di battaglia nella Guerra dei nuclei sono scritti in un linguaggio specializzato che ho chiamato Redcode, molto vicino a quella classe di linguaggi di programmazione detti linguaggi di assemblatore. Molti programmi per calcolatore, oggi, vengono scritti in linguaggi di alto livello come il Pascal, il Fortran o il BASIC; in questi linguaggi un singolo enunciato può specificare un'intera sequenza di istruzioni macchina. Inoltre i singoli enunciati sono facili da leggere e da capire per il programmatore. Perché un programma possa essere eseguito, tuttavia, deve prima essere tradotto in «linguaggio macchina», in cui ciascuna istruzione viene espressa da una lunga successione di cifre binarie. Scrivere un programma in tale forma è quanto meno noioso.

I linguaggi di assemblatore occupano una posizione intermedia tra i linguaggi di alto livello e il linguaggio macchina. In un programma in linguaggio di assemblatore ciascun enunciato corrisponde in genere a una singola istruzione e quindi a una particolare successione di cifre binarie. Il programmatore, tuttavia, invece di scrivere i numeri binari, li rappresenta come brevi parole o abbreviazioni dette mnemoniche (perché sono più facili da ricordare dei numeri). La traduzione nel linguaggio macchina viene operata da un programma detto assemblatore.

Si programma relativamente poco in linguaggi di assemblatore perché i programmi risultanti sono più lunghi e più difficili da capire o modificare dei loro corrispettivi in linguaggio di alto livello. Vi sono tuttavia compiti per i quali un linguaggio di assemblatore è l'ideale. Un programma viene generalmente scritto in un linguaggio di assemblatore quando deve occupare il minor spazio possibile o girare il più rapidamente possibile. Inoltre con un linguaggio di assemblatore si possono fare cose impossibili con un linguaggio di alto livello. Per esempio, un programma in linguaggio di assemblatore può modificare le proprie istruzioni o spostarsi in una nuova posizione di memoria.

La Guerra dei nuclei è stata ispirata da una storia che sentii raccontare alcuni anni fa a proposito di un astuto programmatore del laboratorio di ricerca di una società che chiamerò *X*. Quel programmatore aveva scritto un programma in linguaggio di assemblatore, che aveva chiamato Creeper (ovvero: il Rampicante), in grado di duplicarsi ogni volta che veniva fatto girare. Si poteva anche estendere da un calcolatore all'altro della rete della società *X*, e non aveva altre funzioni che quella di riprodursi. Poco tempo dopo c'erano tante copie di Creeper da invadere ben più utili programmi e dati. L'infestamento crescente non venne tenuto sotto controllo, finché qualcuno pensò di reagire rispondendo con le stesse armi, cioè scrivendo un secondo programma in grado di duplicarsi, chiamato Reaper (ovvero: il Falciatore) il cui scopo era quello di distruggere copie del Creeper finché non ne rimanessero più e poi distruggersi. Reaper svolse il proprio compito e le cose vennero riportate alla normalità nei laboratori *X*.

La storia era troppo bella per non prestarvi fede, anche se aveva lacune molto evidenti. Ci volle un po' di tempo per capire quello che si era effettivamente verificato e che si nascondeva dietro questo pittoresco episodio. Ne darò un resoconto più avanti; per il momento è sufficiente segnalare che il mio desiderio di credervi si basava sull'affascinante idea di due programmi che si danno battaglia nei bui e silenziosi corridoi della memoria principale di un calcolatore.

L'anno scorso decisi che, anche se la storia non fosse stata vera, qualcosa del genere avrebbe potuto accadere. Impostai una versione iniziale della Guerra dei nuclei e, con l'assistenza di David Jones, uno studente del mio dipartimento all'Università dell'Ontario occidentale, la misi in funzione. Da allora abbiamo perfezionato il gioco fino a portarlo a un livello decisamente interessante.

La Guerra dei nuclei ha quattro componenti principali; una matrice di memoria di 8000 indirizzi, il linguaggio di assemblatore Redcode, un programma esecutivo detto MARS (un acronimo di Memory Array Redcode Simulator) e l'insieme dei programmi che si danno battaglia. Due programmi di battaglia sono inseriti nella matrice di memoria in posizioni scelte a caso; nessuno dei due programmi sa dove si trova l'altro. MARS esegue i programmi seguendo una semplice versione della partizione di tempo, una tecnica per distribuire le risorse di un calcolatore tra numerosi utenti. I due programmi si alternano: viene eseguita una istruzione del primo, poi una del secondo e così via.

Spetta esclusivamente al programmatore stabilire che cosa faccia un programma di battaglia durante l'esecuzione dei cicli assegnatigli. L'obiettivo è, ovviamente, quello di distruggere l'altro programma sabotando le sue istruzioni. È anche possibile una strategia difensiva: un programma potrebbe decidere di riparare un danno che ha ricevuto o di portarsi fuori tiro quando è sottoposto a un attacco. La battaglia si conclude quando MARS giunge a un'istruzione ineseguibile in uno dei programmi. Il programma che contiene l'istruzione difettosa (presumibilmente, vittima di guerra) viene dichiarato perdente.

Si può imparare molto di un programma di battaglia semplicemente analizzando le sue azioni mentalmente o con carta e matita. Per sottoporre il programma alla verifica dell'esperienza bisogna però avere accesso a un calcolatore.

I programmi potrebbero anche girare su un calcolatore personale e, a questo proposito, Jones e io abbiamo preparato delle note per coloro che desiderassero impostare per proprio conto una guerra dei nuclei.

Suggerisco pertanto la consultazione di *Linee guida per la Guerra dei nuclei* riportate nell'appendice a questo volume a pagina 147.

Prima di descrivere Redcode e di spiegare alcuni programmi di battaglia, vorrei aggiungere qualche cosa sulla matri-

ce di memoria. Ho detto che consiste di 8000 indirizzi, ma non c'è niente di magico in questo numero: una matrice più piccola potrebbe funzionare altrettanto bene. La matrice di memoria è diversa dalla maggioranza delle memorie di calcolatore per la sua configurazione circolare: è una successione di indirizzi numerati da 0 a 7999, ma da questo punto in avanti si ritorna da capo in modo che l'indirizzo 8000 è equivalente all'indirizzo 0. MARS riduce sempre un indirizzo superiore a 7999 considerando il resto che si ottiene dividendolo per 8000. Se un programma di battaglia ordina un tiro all'indirizzo 9378, MARS interpreta l'indirizzo come 1378.

Redcode è un linguaggio simile a un linguaggio di assemblatore semplificato e di uso specifico. Ha istruzioni per spostare i contenuti di un indirizzo della memoria, a un altro indirizzo, per modificare aritmeticamente i contenuti e per trasferire il controllo avanti e indietro all'interno di un programma. Mentre l'uscita di un assemblatore vero e proprio è costituita da codici binari, la forma mnemonica di un'istruzione in Redcode viene tradotta da MARS in un grande numero intero in notazione decimale, che viene quindi immagazzinato nella matrice di memoria: ogni indirizzo della matrice può contenere un intero cosiffatto. È sempre MARS che interpreta gli interi come istruzioni ed esegue le operazioni indicate.

Un elenco delle istruzioni di base di Redcode è riportato nella figura in alto a pagina 28. Per ciascuna istruzione, il programmatore deve fornire almeno un argomento e molte istruzioni hanno due argomenti. Per esempio, nell'istruzione JMP −7 la mnemonica JMP (che sta per *jump*, salto) è seguita da un solo argomento, −7. L'istruzione dice a MARS di trasferire il controllo all'indirizzo di memoria sette posti prima dell'attuale, cioè sette posti prima della stessa istruzione JMP −7. Se l'istruzione fosse, per esempio, all'indirizzo 3715, l'esecuzione del programma tornerebbe in questo modo all'indirizzo 3708.

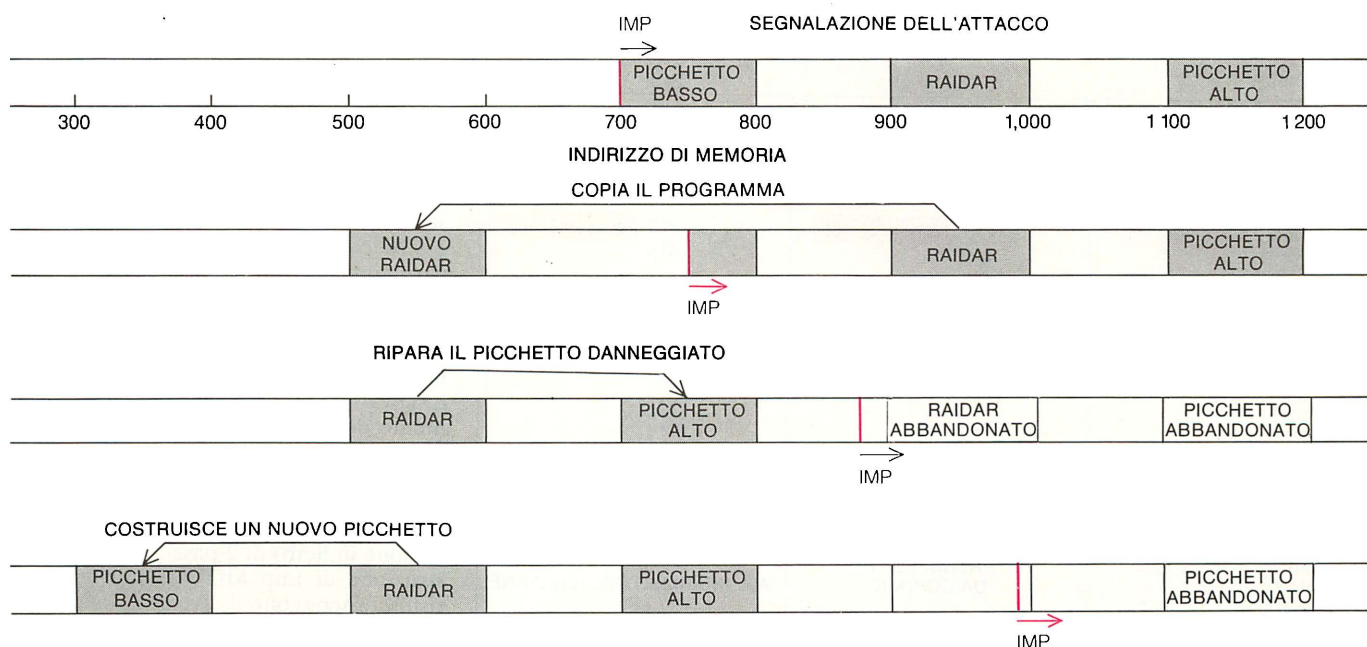
Questo metodo per calcolare una posizione nella memoria viene chiamato indirizzamento relativo ed è l'unico metodo impiegato in Redcode. Un programma di battaglia non ha modo di conoscere la propria posizione assoluta nella matrice di memoria.

L'istruzione MOV 3 100 dice a MARS di avanzare di tre indirizzi, di copiare ciò che vi trova e lasciarlo 100 indirizzi oltre l'istruzione MOV, sovrapponendolo a ciò che vi si trovava. Gli argomenti di questa istruzione vengono dati in modo «diretto», cioè devono essere interpretati come indirizzi su cui operare direttamente. Sono ammessi due altri modi. Far precedere un argomento da una @ lo rende «indiretto». Nell'istruzione MOV @ 3 100 l'intero da lasciare all'indirizzo relativo 100 non è quello trovato all'indirizzo relativo 3 bensì quello trovato all'indirizzo indicato dai contenuti dell'indirizzo relativo 3. (La figura in basso della pagina seguente illustra più in dettaglio il processo di indirizzamento indiretto.) Il segno # rende un argomento «immediato»: l'argomento cioè viene trattato non come un indirizzo, ma come un intero. L'istruzione MOV # 3 100 fa sì che l'intero 3 sia spostato all'indirizzo relativo 100.

La maggior parte delle altre istruzioni non hanno bisogno di ulteriore spiegazione, ma l'enunciato dati (DAT) richiede un commento. Può fungere da spazio di lavoro per contenere le informazioni di cui un programma può avere bisogno. Propriamente parlando, tuttavia, non è un'istruzione; in realtà qualsiasi locazione di memoria con uno zero come sua prima cifra decimale può essere considerato un enunciato DAT e come tale non è eseguibile. Se a MARS venisse richiesto di eseguire una tale «istruzione», non sarebbe in grado di farlo e dichiarerebbe quel programma perdente.

L'intero decimale che codifica una istruzione in Redcode ha parecchi campi o aree funzionali *(si veda la figura centrale della pagina seguente)*. La prima cifra rappresenta la mnemonica e due altre cifre specificano la modalità di indirizzamento (diretto, indiretto o immediato). Inoltre, vengono previste quattro cifre per ogni argomento. Gli argomenti negativi vengono immagazzinati in forma di complemento: −1 sarebbe rappresentato come 7999, dato che nella matrice di memoria circolare aggiungere 7999 ha lo stesso effetto che sottrarre 1.

Nell'illustrazione in alto a pagina 29 sono elencate le istruzioni che compongono un programma di battaglia detto Dwarf. Dwarf è un programma molto stupido, ma molto pericoloso, che si apre la strada attraverso la matrice di memoria bombardando un indirizzo ogni cinque con uno zero. Zero è l'intero che indica un enunciato dati non eseguibile e quindi infilare uno zero in un programma avversario può farlo arrestare.

Poniamo che Dwarf occupi gli indirizzi assoluti da 1 a 4. L'indirizzo 1 contiene inizialmente DAT −1, ma l'esecuzione comincia con l'istruzione successiva ADD # 5 −1. L'effetto dell'istruzione è aggiungere 5 ai contenuti dell'indirizzo precedente, e cioè l'enunciato DAT −1, trasformandolo quindi in DAT 4. Successivamente Dwarf esegue l'istruzione all'indirizzo assoluto 3, MOV # 0 @ −2. Qui l'intero da spostare è 0, specificato come valore immediato. L'indirizzo bersaglio viene calcolato indirettamente nel modo seguente. Prima MARS torna indietro di due indirizzi a partire dall'indirizzo 3, arrivando all'indirizzo 1. Poi esamina il valore dei dati in quel punto, nel caso

Raidar, un raffinato programma di battaglia, sfugge al più semplice Imp nella matrice di memoria della Guerra dei nuclei

ISTRUZIONE	MNEMONICA	CODICE	ARGOMENTI		SPIEGAZIONE
Trasferimento	MOV	1	A	B	Trasferisci i contenuti dell'indirizzo *A* all'indirizzo *B*.
Somma	ADD	2	A	B	Somma i contenuti dell'indirizzo *A* all'indirizzo *B*.
Sottrazione	SUB	3	A	B	Sottrai i contenuti dell'indirizzo *A* dall'indirizzo *B*.
Salto	JMP	4	A		Trasferisci il controllo all'indirizzo *A*.
Salto se zero	JMZ	5	A	B	Trasferisci il controllo all'indirizzo *A* se i contenuti dell'indirizzo *B* sono zero.
Salto se maggiore	JMG	6	A	B	Trasferisci il controllo all'indirizzo *A* se i contenuti di *B* sono maggiori di zero.
Decremento: salto se zero	DJZ	7	A	B	Sottrai 1 dai contenuti dell'indirizzo *B* e trasferisci il controllo all'indirizzo *A* se i contenuti dell'indirizzo *B* sono, a quel punto, zero.
Confronto	CMP	8	A	B	Confronta i contenuti degli indirizzi *A* e *B*; se non sono uguali, salta l'istruzione successiva.
Enunciato di dati	DAT	0		B	Un enunciato non eseguibile; *B* è il valore dei dati.

Le istruzioni di Redcode, un linguaggio di assemblatore per la Guerra dei nuclei

MNEMONICA	ARGOMENTO *A*	ARGOMENTO *B*	CODICE OPERATIVO	CIFRA DEL MODO: ARGOMENTO *A*	CIFRA DEL MODO: ARGOMENTO *B*	ARGOMENTO *A*	ARGOMENTO *B*
DAT		−1	0	0	0	0000	7999
ADD	#5	−1	2	0	1	0005	7999
MOV	#0	@ −2	1	0	2	0000	7998
JMP	−2		4	1	0	7998	0000

MODI DI INDIRIZZAMENTO:		
IMMEDIATO	#	0
DIRETTO		1
INDIRETTO	@	2

La codificazione in interi decimali delle istruzioni di Redcode

412	412	412
413 DAT 22	418 − 5 413 DAT 22	413 DAT 22
414	414	414
415 MOV @3 100	415 MOV @3 100	415 MOV @3 100
416	416	416
417	417	417
415 + 3 418 DAT −5	418 DAT −5	418 DAT −5
419	419	419
420	420	420
• • •	• • •	• • •
514	514	514
515	515	415 + 100 515 DAT 22
516	516	516
PRENDI L'INDIRIZZO SORGENTE	PRENDI I DATI DA COPIARE	COPIA I DATI ALL'INDIRIZZO DESTINAZIONE

Il meccanismo a tre passi dell'indirizzamento relativo indiretto

specifico 4, e lo interpreta come un indirizzo relativo alla posizione attuale; in altre parole conta quattro posti in avanti a partire dall'indirizzo 1 e quindi deposita uno 0 all'indirizzo 5.

L'istruzione finale del Dwarf, JMP −2, crea un ciclo senza fine. Riporta indietro l'esecuzione all'indirizzo assoluto 2, che di nuovo incrementa l'enunciato DAT di 5: il suo nuovo valore sarà DAT 9. Nel successivo ciclo di esecuzione viene pertanto inviato uno 0 all'indirizzo assoluto 10. Le successive bombe 0 cadranno negli indirizzi assoluti 15, 20, 25 e così via. Il programma in se stesso è immobile, ma la sua artiglieria minaccia l'intera matrice. Alla fine Dwarf si apre la strada fino agli indirizzi 7990, 7995 e poi 8000. Per quanto riguarda MARS, 8000 è uguale a 0 e quindi Dwarf ha evitato per poco il suicidio. Il suo missile successivo atterra di nuovo all'indirizzo 5.

È bene rendersi conto che nessun programma di battaglia stazionario che abbia più di quattro istruzioni può evitare di essere colpito da Dwarf. Il programma avversario ha solo tre scelte: spostarsi e quindi eludere il bombardamento, incassare i colpi e riparare il danno oppure raggiungere Dwarf per primo. Per avere successo in quest'ultima strategia può darsi che il programma debba affidarsi alla fortuna: può non avere nessuna idea del punto della matrice di memoria in cui Dwarf si trova e, in media, ha circa 1600 cicli di esecuzione prima che un colpo vada a segno. Se anche il secondo programma è un Dwarf, ciascuno di essi vince il 30 per cento delle volte. Nel 40 per cento dei casi nessuno dei due programmi arriva a lanciare il colpo finale.

Prima di prendere in considerazione le altre due strategie, vorrei presentare un curioso programma di battaglia di una sola riga, che noi chiamiamo Imp. Eccolo:

MOV 0 1

Imp è l'esempio più semplice di programma in Redcode in grado di ricollocarsi nella matrice di memoria. Trascrive i contenuti dell'indirizzo relativo 0 (e precisamente MOV 0 1) nell'indirizzo relativo 1, cioè nell'indirizzo successivo. Quando il programma viene eseguito, si muove lungo la matrice a una velocità di un indirizzo per ciclo, lasciando dietro di sé una scia di istruzioni MOV 0 1.

Che cosa succede se mettiamo Imp in gara contro Dwarf? Lo sbarramento di zero disposto da Dwarf muove attraverso la matrice di memoria più velocemente di Imp, ma non ne segue necessariamente che Dwarf sia in vantaggio. La domanda è: Dwarf colpirà Imp anche se lo sbarramento dovesse raggiungerlo?

Se Imp raggiunge Dwarf per primo, probabilmente si aprirà un varco attraverso il codice di Dwarf e, quando l'istruzione di Dwarf JMP − 2 trasporterà l'esecuzione indietro di 2 passi, vi troverà l'istruzione di Imp MOV 0 1. Dwarf ne risulterà sconvolto e diventerà un secondo Imp che inseguirà senza fine il primo lungo la matrice. Secondo le regole della Guerra dei nuclei la partita si conclude

INDIRIZZO	CICLO 1	CICLO 2	CICLO 3
0			
1	DAT −1	DAT 4	DAT 14
2	ADD #5 −1	ADD #5 −1	ADD #5 −1
3	MOV #0 @ −2	MOV #0 @ −2	MOV #0 @ −2
4	JMP −2	JMP −2	JMP −2
5		— 0	— 0
6			
7			
8			
9			— 0
10			
11			
12			
13			
14			— 0
15			
16			
17			

Dwarf, un programma di battaglia, deposita uno sbarramento di «bombe zero»

con un pareggio. (Si noti che questo è il risultato che ci si può aspettare «con tutta probabilità». Invito i lettori ad analizzare altre possibilità e magari a scoprire il bizzarro risultato di una di esse.)

Sia Imp che Dwarf rappresentano una classe di programmi che si possono definire piccoli e aggressivi, ma non intelligenti. Al livello successivo stanno programmi che sono più ampi e un po' meno aggressivi, ma abbastanza astuti da affrontare i programmi della classe inferiore. I programmi più astuti hanno la capacità di vanificare un attacco trascrivendo se stessi in una posizione fuori pericolo. Ogni programma del genere include un segmento di codice che assomiglia a quello chiamato Gemini che è riportato nell'illustrazione in basso di questa pagina. Gemini non è stato pensato come un programma di battaglia completo. La sua sola funzione è quella di produrre una copia di se stesso 100 indirizzi più avanti rispetto alla sua posizione attuale e quindi di trasferire l'esecuzione alla nuova copia.

Il programma Gemini ha tre parti principali. Due enunciati di dati all'inizio fungono da puntatori: indicano l'istruzione successiva da copiare e la sua destinazione. Un ciclo a metà del programma opera l'effettiva trascrizione spostando ciascuna istruzione a turno a un indirizzo 100 posti oltre la sua posizione in quel momento. A ogni esecuzione del ciclo entrambi i puntatori vengono aumentati di 1, venendo così a indicare un nuovo indirizzo di partenza e di destinazione. L'istruzione di confronto *compare* (CMP) all'interno del ciclo controlla il valore del primo enunciato di dati; quando è stato aumentato nove volte, l'intero programma è stato trascritto e si esce dal ciclo. Rimane da dare un aggiustamento finale. L'indirizzo destinazione è il secondo enunciato del programma e ha un valore iniziale DAT 99; quando viene trascritto, tuttavia, è già stato incrementato una volta, cosicché nella nuova versione del programma si legge DAT 100. Questo errore di trascrizione viene corretto (dall'istruzione MOV # 99 93), poi l'esecuzione viene affidata alla nuova copia.

Modificando Gemini è possibile creare un'intera classe di programmi di battaglia. Uno di questi, Juggernaut, si trascrive 10 locazioni in avanti, anziché 100. Come Imp cerca di passare attraverso ogni opposizione. Vince molto più spesso di Imp, tuttavia, e pareggia un numero minore di volte perché è meno probabile che un programma sovrascritto sia in grado di eseguire frammenti del codice di Juggernaut. Bigfoot, un altro programma che impiega il meccanismo di Gemini, tiene come intervallo tra le copie un grande numero primo. Bigfoot è difficile da intercettare e ha lo stesso effetto devastante di Juggernaut sull'avversario.

Né Bigfoot né Juggernaut sono molto intelligenti. Finora abbiamo scritto solo due programmi di battaglia che hanno le caratteristiche per essere classificati in un secondo livello di raffinatezza. Sono troppo lunghi per poter essere riportati in questa sede. Uno di essi, che chiamiamo Raidar, ha due «picchetti» ai due lati del programma (*si veda l'illustrazione a pagina 27*). Ciascun picchetto è composto da 100 indirizzi consecutivi pieni di 1 ed è separato dal programma da una zona cuscinetto di 100 indirizzi vuoti. Raidar divide il suo tempo tra attacchi sistematici ad aree distanti della matrice di memoria e il controllo degli indirizzi dei suoi picchetti. Se uno dei picchetti risulta modificato, Raidar interpreta il cambiamento come prova di un attacco da parte di Dwarf, Imp o qualche altro programma non intelligente. Raidar allora si trascrive dall'altra parte del picchetto danneggiato, lo ripara, costruisce un nuovo picchetto sul lato rimasto scoperto e riprende il funzionamento normale.

A un programma di battaglia si può dare, oltre a quella di trascrivere se stesso, la capacità di effettuare la manutenzione di se stesso. Jones ha scritto un programma «ad automanutenzione» che può sopravvivere ad alcuni attacchi, anche se non a tutti. È chiamato Scanner e conserva due copie di se stesso, ma generalmente ne pone in esecuzione solo una. La copia che viene fatta girare analizza periodicamente l'altra copia per vedere se una delle sue istruzioni è stata modificata da un attacco. Si individuano i cambiamenti confrontando le due copie, ammettendo sempre che la copia in esecuzione sia corretta. Se vengono trovate istruzioni errate, vengono sostituite e il controllo è trasferito all'altra copia che comincia allora ad analizzare la prima.

Fino a questo punto Scanner rimane un programma puramente di difesa: è capace di sopravvivere agli attacchi di Dwarf, Imp, Juggernaut e analoghi aggressori dal movimento lento, almeno se l'attacco viene sferrato a partire dalla direzione giusta. Jones sta attualmente lavorando a un programma ad automanutenzione che conserva tre copie di se stesso.

Sono curioso di vedere se i lettori sapranno ideare altri tipi di programmi ad automanutenzione. Per esempio, si potrebbe pensare di riparare due o più copie di un programma anche se solo una copia viene messa in esecuzione. Il programma potrebbe comprendere una sezione di manutenzione che potrebbe servirsi di una copia extra mentre ripara le istruzioni danneggiate. La sezione di manutenzione potrebbe riparare anche se stessa, ma potrebbe ancora essere vulnerabile, in alcune posizioni. Una misura di vulnerabilità assume che sia stata colpita una sola istruzione; in media, quante istruzioni devono venire colpite per arrivare alla

	DAT		0	/puntatore all'indirizzo sorgente
	DAT		99	/puntatore all'indirizzo destinazione
CICLO	MOV	@ −2	@ −1	/copia la sorgente alla destinazione
CICLO	CMP	−3	#9	/se sono state copiate tutte le righe...
CICLO	JMP	4		/...allora lascia il ciclo;
CICLO	ADD	#1	−5	/altrimenti, incrementa l'indirizzo sorgente
CICLO	ADD	#1	−5	/...e l'indirizzo destinazione...
CICLO	JMP	−5		/...e ritorna al ciclo
	MOV	#99	93	/ristabilisci l'indirizzo destinazione iniziale
	JMP	93		/salta alla nuova copia

Gemini, un programma che copia se stesso in una nuova posizione della matrice di memoria

7978	MOV	0	1	
7979	MOV	0	1	
7980	—	0		
7981	MOV	0	1	
7982	MOV	0	1	
7983	MOV	0	1	
7984	MOV	0	1	
7985	—	0		
7986	MOV	0	1	
7987	MOV	0	1	
7988	MOV	0	1	
7989	MOV	0	1	
7990	—	0		
7991	MOV	0	1	
7992	MOV	0	1	
7993	MOV	0	1	
7994	MOV	0	1	} IMP
7995				
7996				
7997				
7998				
7999				
0				
1	DAT		7994	DWARF
2	ADD	#5	−1	
3	MOV	#0	@ −2	
4	JMP	−2		
5	—	0		
6				
7				
8				
9				
10	—	0		
11				

Imp contro Dwarf: chi vince?

definitiva morte del programma? Secondo questo criterio di misurazione, qual è il programma ad automanutenzione meno vulnerabile che si possa scrivere?

Solo se si possono elaborare programmi ragionevolmente robusti la Guerra dei nuclei raggiungerà il livello di un gioco eccitante in cui l'accento cada più sull'attacco che sulla difesa. I programmi di battaglia allora dovranno scovare e identificare il codice nemico e sferrare un intenso attacco ovunque venga individuato.

Posso aver dato l'impressione che Redcode e l'intero sistema MARS siano fissi, ma non è così. Nei momenti liberi abbiamo sperimentato nuove idee e siamo certamente aperti a ogni suggerimento. A dir la verità, abbiamo passato tanto tempo a sperimentare nuovi programmi e nuove caratteristiche che alcune battaglie non sono mai state combattute nel nostro sistema.

Un'idea con cui abbiamo lavorato è quella di aggiungere un'istruzione extra che renda un po' più facile l'automanutenzione o l'autoprotezione. L'istruzione PCT *A* proteggerebbe *(protect)* l'istruzione all'indirizzo *A* da ogni modifica fino alla successiva esecuzione. Di quanto sarebbe ridotta la vulnerabilità di un programma con un'istruzione di questo tipo?

Nelle note di cui si è parlato in precedenza, non solo descriviamo le regole della Guerra dei nuclei ma anche come impostare una matrice di memoria e scrivere un sistema MARS in vari linguaggi di alto livello. Diamo anche indicazioni su come visualizzare i risultati. Per ora le regole seguenti definiscono il gioco con sufficiente precisione da permettere ai giocatori di ideare con carta e matita programmi di battaglia:

1. I due programmi di battaglia vengono caricati nella matrice di memoria in posizioni casuali, ma all'inizio debbono essere separati da almeno 1000 indirizzi.

2. MARS esegue alternativamente un'istruzione di ciascun programma finché non raggiunge un'istruzione che non può essere eseguita. Il programma con l'istruzione errata perde.

3. I programmi possono essere attaccati con qualsiasi arma disponibile. Una «bomba» può essere rappresentata da uno 0 o da un altro intero qualsiasi, compresa un'istruzione in Redcode valida.

4. Viene fissato un limite di tempo all'incontro, determinato dalla velocità del calcolatore. Se si raggiunge il limite di tempo ed entrambi i programmi stanno ancora girando, si ha un pareggio.

La storia di Creeper e Reaper sembra basata sulla composizione di due programmi esistenti, il primo un gioco chiamato Darwin, inventato da M. Douglas McIlroy degli AT&T Bell Laboratories e il secondo, chiamato Worm, scritto da John F. Shoch dello Xerox Research Center di Palo Alto. Entrambi i programmi hanno alcuni anni e c'è stato quindi tempo perché le voci si propagassero e la storia si arricchisse. (Darwin è stato descritto in *Software: Practice and Experience,* volume 2, pagine 93-96, 1972. Una descrizione vaga di quello che sembra lo stesso gioco si trova anche nell'edizione 1978 di *Computer Lib.*)

Nel Darwin ogni giocatore mette in campo un certo numero di programmi in linguaggio di assemblatore (gli «organismi») i quali coabitano nella memoria principale con gli organismi dell'altro giocatore. Gli organismi creati da un giocatore (e quindi appartenenti alla stessa «specie») tentano di uccidere quelli dell'altra specie e di occupare il loro spazio. Vincitore del gioco è quel giocatore che ha più organismi quando scade il tempo. McIlroy inventò un organismo che non si poteva uccidere, anche se vinceva solo «alcune volte». Era immortale, ma, evidentemente, non molto aggressivo.

Worm era un programma sperimentale ideato per fare il maggior uso possibile di minicalcolatori collegati in rete alla Xerox. Worm veniva caricato in macchine in stato di riposo da un programma supervisore. Il suo scopo era quello di prendere il controllo della macchina e, in collaborazione con altri Worm insediati in altre macchine in stato di riposo, far girare grandi programmi applicativi nel sistema multielaboratore risultante. Worm era concepito in modo che chiunque volesse usare una delle macchine occupate poteva farlo agevolmente senza interferire con il lavoro più ampio.

Si possono trovare elementi sia di Darwin sia di Worm nella storia di Creeper e Reaper. Nella Guerra dei nuclei Reaper è divenuto realtà.

Soluzioni proposte

Douglas McIlroy mi ha detto che l'inventore del gioco Darwin è Victor A. Vyssotsky. McIlroy, invece, ha inventato un organismo che non si può uccidere.

Che cosa avviene quando un Imp si fonde con Dwarf? Una possibilità è già stata spiegata: Dwarf trasferisce il controllo al codice di Imp e diviene un secondo Imp che rincorre senza fine il primo. Un altro possibile esito ha l'effetto opposto. Supponiamo che Dwarf sia appena saltato indietro alla sua prima istruzione quando Imp si copia nella locazione dei dati di Dwarf. La situazione è allora:

```
Imp → MOV    0     1
Dwarf → ADD  #5   −1
        MOV  #0   @ −2
        JMP  −2
```

Dato che è il turno di Dwarf, esso somma 5 al codice di Imp, trasformandolo in MOV 0 6. Poi è il turno di Imp, che si copia sei spazi avanti, distante da Dwarf che poi bombarda il suo indirizzo successivo (specificato dal codice numerico corrispondente a MOV 0 6). Al turno successivo, Imp esegue la prima linea del programma di Dwarf e per una volta la partita è giocata da un «doppio Dwarf» che spara inutilmente sulla matrice di nuclei mentre l'oggetto del suo attacco occupa il suo stesso corpo e fa la stessa cosa!

David Menconi, un progettista di giochi dell'Atari, Inc., ha suggerito di rendere questo fenomeno una caratteristica regolare della Guerra dei nuclei, ammettendo che ogni programma di battaglia possa andare in esecuzione in due posti alla volta. Così se un programma perde un «se stesso», un altro se stesso potrebbe essere in grado di riparare il danno.

Finora il sistema più efficace è stato costruito da tre studenti universitari: Gordon J. Goetsch e Michel L. Mauldin della Carnegie-Mellon University e Paul G. Milazzo della Rice University. Mauldin ha dato una dimostrazione del programma su un calcolatore VAX. L'intera matrice di nuclei appare sullo schermo, con la posizione di ciascun programma in gara indicata da una lettera maiuscola e le aree interessate dal programma indicate dalle corrispondenti lettere minuscole. Mauldin ha inventato un programma di battaglia che opera come Dwarf tranne per il fatto che le sue bombe sono dirette secondo la successione dei numeri di Fibonacci (1, 1, 2, 3, 5 e così via, dove ogni numero è la somma dei due precedenti). Stranamente, Dwarf batte Mortar il 60 per cento delle volte, ma Mortar uccide sempre un programma ad automanutenzione in tre parti chiamato Voter. Voter, invece, sopravvive agli attacchi di Dwarf e lo batte regolarmente. Goetsch, Mauldin e Milazzo hanno concluso che se un programma è più lungo di 10 istruzioni dev'essere ad automanutenzione per sconfiggere Mortar, ma nessun programma può sopravvivervi se è più lungo di 141 istruzioni.

Un bestiario di virus, bachi e altre insidie per la memoria dei calcolatori nella Guerra dei nuclei

di A. K. Dewdney

Le Scienze, maggio 1985

La mia descrizione di programmi in linguaggio macchina che si aggirano nella memoria di un calcolatore cercando di distruggersi l'un l'altro ha avuto notevole risonanza. Secondo molti resoconti, abbondano gli esempi di bachi, virus e altre creature software annidate in tutti i possibili ambienti di elaborazione. Alcune possibilità sono così orribili che esito a riportarle.

Il romanzo francese di spionaggio *Softwar: La Guerre Douce* presenta un'immaginaria situazione geopolitica di questo genere. Gli autori, Thierry Breton e Denis Beneich, imbastiscono un agghiacciante racconto a proposito dell'acquisto, da parte dell'Unione Sovietica, di un supercalcolatore americano. Invece di bloccare la vendita, le autorità americane acconsentono alla transazione mostrando una studiata riluttanza: il calcolatore, infatti, è stato segretamente programmato con una «bomba software». Ufficialmente acquistata per le previsioni meteorologiche sul vasto territorio dell'Unione Sovietica, la macchina, o meglio il suo software, contiene un «grilletto» nascosto: appena il Servizio meteorologico nazionale degli Stati Uniti comunica il rilevamento di una certa temperatura a St. Thomas, nelle Virgin Islands, il programma procede a sovvertire e distruggere tutti i pezzi di software che riesce a trovare nella rete sovietica. Se è vero che sceneggiature di questo genere rappresentano possibilità reali, sono tentato di dire: «Se guerra [*war*] deve essere, che sia almeno dolce [*soft*]». D'altra parte, un dubbio mi viene dalla possibilità di un incidente dovuto al collegamento stretto fra software militare e sistemi di controllo delle armi.

Prima di passare a descrivere le esperienze avute da vari lettori con programmi ostili, vale senz'altro la pena di riassumere le caratteristiche principali della Guerra dei nuclei già fornite nell'articolo precedente.

Due giocatori scrivono ciascuno un programma in un linguaggio di basso livello chiamato REDCODE. I programmi vengono poi posti in un'ampia arena circolare che chiamiamo Nucleo: in realtà nient'altro che una matrice di migliaia di locazioni, in cui l'ultimo indirizzo è contiguo al primo. Ogni istruzione del programma da battaglia occupa una locazione nel Nucleo. Il programma esecutivo MARS (acronimo per *Memory Array Redcode Simulator*, ovvero «simulatore di Redcode nella matrice di memoria») fa girare i programmi da battaglia eseguendo alternativamente un'istruzione dell'uno e una dell'altro, come un semplice sistema a partizione di tempo: i due programmi si attaccano e, a turno, cercano di evitare danni o di riparare quelli subiti. Una semplice modalità d'attacco può essere eseguita per mezzo di istruzioni MOV. Per esempio:

MOV # 0 1000

fa sì che il numero 0 sia posto nella locazione il cui indirizzo si trova 1000 locazioni al di là di questa istruzione, cancellando il precedente contenuto di quella locazione. Nel caso lo 0 venisse posto su un'istruzione dell'avversario, anch'essa sarebbe tolta di mezzo e il programma non sarebbe più eseguibile: l'avversario avrebbe perso il gioco.

Dato che nessun calcolatore, personale o *mainframe*, è dotato all'origine di REDCODE e di un'adeguata matrice da battaglia, queste caratteristiche devono essere simulate.

Ispirandosi a un articolo di L. S. Penrose sui meccanismi che si autoriproducono, apparso su «Scientific American» nel giugno 1959, Frederick G. Stahl di Chesterfield, Missouri, ha creato un universo lineare in miniatura in cui umili creature vivono, si muovono e (in un certo senso) compiono il proprio destino. Scrive Stahl:

«Come nella Guerra dei nuclei, ho isolato un segmento lineare chiuso di memoria principale in cui una creatura era simulata dal linguaggio macchina modificato. La macchina era un IBM Tipo 650 con memoria a támburo. La creatura era programmata per strisciare lungo il suo universo mangiando cibo (parole diverse da zero) e creando un duplicato di se stessa quando era stato accumulato abbastanza cibo. Come nella Guerra dei nuclei, avevo un programma esecutivo che teneva conto di chi era vivo e distribuiva il tempo d'esecuzione tra le creature viventi. Lo chiamavo «la mano sinistra di Dio». Stahl prosegue analizzando la capacità del suo

ISTRUZIONE	MNEMONICA	CODICE	ARGOMENTI	SPIEGAZIONE
Sposta	MOV	1	A B	Sposta il contenuto dell'indirizzo *A* all'indirizzo *B*
Somma	ADD	2	A B	Somma i contenuti dell'indirizzo *A* e dell'indirizzo *B*
Sottrae	SUB	3	A B	Sottrae il contenuto dell'indirizzo *A* dall'indirizzo *B*
Salta	JMP	4	A	Trasferisce il controllo all'indirizzo *A*
Salta se zero	JMZ	5	A B	Trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* è zero
Salta se maggiore	JMG	6	A B	Trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* è maggiore di zero
Decremento: salta se zero	DJZ	7	A B	Sottrae 1 dal contenuto dell'indirizzo *B* e trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* diventa zero
Confronta	CMP	8	A B	Confronta i contenuti degli indirizzi *A* e *B*; se sono diversi, salta l'istruzione successiva
Divide	SPL	9	A	Divide l'esecuzione nell'istruzione successiva e nell'istruzione in *A*
Enunciato di dati	DAT	0	B	Enunciato non eseguibile; *B* è il valore dei dati

Elenco di istruzioni per la Guerra dei nuclei

```
1   IF PEEK (104) = 134 GOTO 10
2   POKE 104, 134: POKE 134 * 256,0
3   PRINT CHR$(4) "RUN APPLE WORM"
10  HOME : POKE - 16302,0: POKE - 16304,0: POKE 1023,160
20  FOR I = 0 TO 94: READ D: POKE 1024 + I, D: NEXT I
30  POKE - 16368,0
40  IF PEEK ( - 16384)<128 GOTO 40
50  CALL 1024
100 DATA 160,225,200,185,255,3,153,127,4,192,95,208,245,
    160,18,190,76,4,24,189,128,4,105,128,157,128,4,189,129,
    4,105,0,157,129,4,192,13,208,18,238,23,4,173,23,4
200 DATA 141,151,4,206,31,4,173,31,4,141,159,4,136,208,211,
    173,167,4,72,173,176,4,141,167,4,104,141,176,4,76,128,
    4,7,20,25,28,33,46,55,61,65,68,72,75,4,16,40,43,49,52
```

Un baco che vive negli Apple

programma di riprodursi e descrivendo un interessante meccanismo di mutazione: un programma potrebbe subire, durante la copiatura, un piccolo numero di cambiamenti casuali nel suo codice. Tuttavia, riferisce Stahl, «abbandonai questa linea di lavoro dopo una sessione di produzioni in cui un mutante sterile uccise e mangiò l'unica creatura feconda dell'universo. Era chiaro che per ottenere qualche risultato interessante sarebbero state necessarie memorie incredibilmente grandi e tempi di elaborazione lunghissimi.»

Una storia analoga riguarda un gioco chiamato Animale, in cui un programma cerca di stabilire a che animale sta pensando un uomo. David D. Clark, del Laboratory for Computer Science del Massachusetts Institute of Technology, scrive che gli impiegati di una certa azienda giocavano con vero ardore ad Animale. Pur non somigliando a un programma di battaglia e nemmeno alle semplici creature di Stahl, Animale aveva la capacità di riprodursi nei corridoi del nucleo sfruttando gli sforzi del programmatore di potenziare una caratteristica chiave del gioco: quando sbaglia nell'indovinare l'animale che il giocatore umano ha in mente, il programma chiede a quest'ultimo di suggerire una domanda che esso potrebbe porre per migliorare le sue prestazioni future. Questa caratteristica, prosegue Clark, portò il programmatore a inventare un trucco per assicurarsi che ognuno avesse sempre la stessa versione di Animale.

«Su un sistema di elaborazione antiquato, privo di una struttura di catalogo condivisa e privo anche di mezzi di protezione, un programmatore inventò un modo molto originale per rendere disponibile il gioco a più utenti. Una versione del gioco esisteva nel catalogo degli archivi di un utente; quando questi giocava, il programma produceva una copia di se stesso in un altro catalogo di archivi. Se quel catalogo conteneva già una copia del gioco, la vecchia versione veniva cancellata, così che il comportamento del gioco cambiava in modo inatteso per il giocatore. Se quel catalogo non conteneva in precedenza una versione di Animale, un altro utente si trovava ad avere a disposizione il gioco.»

Clark ricorda che Animale era un gioco talmente popolare che, alla fine, tutti i cataloghi del sistema dell'azienda ne contenevano una copia. «Quando poi degli impiegati dell'azienda venivano trasferiti ad altre divisioni... portavano con sé anche Animale, che così si diffuse di macchina in macchina all'interno dell'azienda.» La situazione non sarebbe mai diventata seria se non fosse stato che tutte quelle copie del gioco, per altri versi innocue, cominciarono a intasare la memoria su disco. Solo quando qualcuno inventò una versione più «virulenta» del gioco, la situazione poté tornare sotto controllo. Quando la nuova versione di Animale veniva giocata, essa si copiava in altri cataloghi non una ma due volte. Dandogli abbastanza tempo, si pensava, questo programma avrebbe alla fine cancellato tutte le vecchie versioni di Animale. Dopo un anno, al raggiungimento di una data prefissata, in ogni copia del nuovo programma Animale si innescò un nuovo meccanismo. «Invece di replicarsi due volte, ora esso giocava una partita finale, augurava "arrivederci" all'utente e poi si cancellava. Così Animale venne espulso dal sistema.»

Una volta Ruth Lewart di Holmdel, New Jersey, creò un mostro (per così dire) senza neanche scrivere un programma. Mentre lavorava, sul sistema a partizione di tempo della sua azienda, alla preparazione di una versione dimostrativa di un programma didattico, decise di produrre una copia di riserva su un altro sistema a partizione di tempo. Quando il sistema originale cominciò ad apparire lento, riferisce, «inserii il sistema ausiliario, che era molto sensibile, per tre minuti interi, durante i quali non ci fu alcuna risposta e il caos più completo regnava sullo schermo del mio terminale grafico. Nessuno era più in grado di inserirsi o di uscire dal sistema. La conclusione che si poteva trarre era una sola: in qualche modo la colpa era del mio programma! Nonostante il panico, mi resi improvvisamente conto di aver usato una "e" commerciale (&) come carattere separatore di campo del terminale. Ma la & era anche il carattere usato dal sistema per generare un processo di fondo. Alla prima lettura dallo schermo, il calcolatore deve aver intercettato le & indirizzate al terminale e deve aver generato un gran numero di processi, ciascuno dei quali, a sua volta, generò altri processi, e così via all'infinito.» Una telefonata frenetica informò un responsabile di sistema dell'origine del disturbo e il calcolatore *mainframe* venne spento e fatto ripartire. Inutile dire che Lewart cambiò la & in un altro carattere e il suo programma «da allora ha sempre funzionato felicemente».

Anche se i programmi per la Guerra dei nuclei non vengono generati in questo modo, copie aggiuntive possono aumentare le loro capacità di sopravvivenza. Parecchi lettori hanno proposto la realizzazione di tre copie del programma, in modo che la copia in esecuzione possa utilizzare le altre due per stabilire se qualche sua istruzione è sbagliata. Il programma in esecuzione potrebbe sostituire un'istruzione difettosa con una idonea a garantire la sopravvivenza. Un'idea analoga sta alla base di Scavenger (Spazzino), un programma ideato per proteggere da errori gli archivi di una memoria di massa quando si preparano copie di riserva su nastro magnetico. Arthur Hudson, che vive a Newton nel Massachusetts (e lavora per un'altra azienda che non citiamo), scrive: «Chiunque abbia usato spesso il nastro magnetico si è trovato assediato da una forza aliena chiamata Legge delle probabilità composte.» Hudson prosegue citando vari errori connessi con l'uso di nastri e dimostra che, sebbene ogni tipo di errore abbia una probabilità relativamente piccola di prodursi, la probabilità che se ne verifichi almeno uno è spiacevolmente alta. Poi continua così:

«Niente paura, Scavenger è con voi: se gli affidate un archivio di una memoria di massa, copierà l'archivio su tre nastri magnetici senza seccarvi con banali dettagli di gestione. Anche se il calcolatore cade in errore logico (come avveniva parecchie volte al giorno) di solito non verrà distrutta la registrazione dell'esecuzione; quando il calcolatore viene riattivato, tutti i bachi presenti nella registrazione si metteranno a loro volta in funzione. Alla scrittura di ciascun nastro presiede un compito distinto, sotto il coordinamento di un programma principale.»

Chi possiede un calcolatore Apple dovrebbe guardarsi da uno spregevole programmino chiamato Apple Worm (letteralmente Baco della mela), creato da Jim Hauser e William R. Buckley della California Polytechnic State University a San Luis Obispo. Scritto per l'Apple II in linguaggio di assemblatore del microelaboratore 6502, questa specie di baco si riproduce nel corso di un allegro viaggetto attraverso l'Apple ospite. All'inizio si carica un particolare programma BASIC *(si veda l'illustrazione della pagina precedente),* il quale a sua volta carica il baco nella parte bassa della memoria (quella con indirizzi bassi). Il programma BASIC occupa invece la parte alta della memoria.

«Dato che il Baco viene caricato in una delle aree grafiche della macchina, si può osservare il Baco che inizia il suo attacco a capofitto (o, per meglio dire, a codafitta) nella memoria alta... Una volta che il Baco ha lasciato la finestra grafica... si può aspettare che abbia cancellato la parte alta della memoria (compreso il programma BASIC) e vada a scontrarsi con le ROM di sistema.»

Hauser e Buckley hanno in programma di pubblicare, in un futuro non lontano, una raccolta di bachi. Hanno progettato un Worm Operating System (Sistema Operativo Baco) e hanno perfino scritto un videogioco in cui Baco ricopre uno dei ruoli principali.

Una minaccia software è stata proposta da Roberto Cerruti e Marco Morocutti di Brescia che hanno pensato a un modo per infestare l'Apple II, non con un baco ma con un virus. Scrive Cerruti: «Marco pensò di scrivere un programma capace di passare da un calcolatore all'altro, come un virus, e di "infettare" in questo modo altri Apple. Non fummo però in grado di idearlo finché non mi resi conto che il programma doveva infettare i dischetti e usare il calcolatore solo come mezzo per migrare da un disco all'altro. Fu così che il nostro virus cominciò a prendere forma.

«Come si sa, ogni dischetto Apple contiene una copia del DOS (il sistema operativo per dischi), che viene lanciato dal calcolatore non appena riceve l'alimentazione. Il virus era una modificazione di questo DOS, che a ogni operazione di scrittura verificava la propria presenza sul disco e, in caso negativo, modificava il DOS sul disco, copiandosi così su ogni dischetto messo nell'unità di lettura e registrazione dopo la prima accensione. Pensammo che, se avessimo installato un simile DOS su alcuni dischi usati nel maggiore negozio di calcolatori della nostra città, Brescia, in breve avremmo provocato la diffusione di un'epidemia in tutta la città.

«Ma era una vera epidemia, con virus così innocui? No, i nostri virus dovevano essere maligni! Decidemmo, quindi, che dopo 16 cicli di autoriproduzione, contati sul disco stesso, il programma dovesse decidere di reinizializzare il disco subito dopo il lancio. A quel punto era chiara la terribile perversità della nostra idea e decidemmo di non metterla in pratica né di parlarne con alcuno.»

Cerruti e Morocutti sono stati gentili. In un calcolatore personale, il sistema operativo per dischi è l'arbitro ultimo del destino dei programmi, dei dati e di tutto il resto. Nello schema appena descritto, il sistema operativo infetto cancella il disco da cui è stato lanciato e quindi non può più essere caricato se non da un altro disco. Il DOS contagiato potrebbe anche far apparire periodicamente un messaggio irritante del tipo:

IL VOSTRO DISCO
VI SFUGGE?
È ora di rivolgersi al
DOTTOR DOS
disponibile su disco nel
più vicino negozio di calcolatori

L'infezione virale descritta si è già verificata su piccola scala. Richard Skrenta, Jr., studente di una scuola superiore di Pittsburgh, scrisse un programma di questo genere. Invece di cancellare dischi o visualizzare avvisi, questa forma di infezione provocava la comparsa di subdoli errori nel sistema operativo.

«Tutto questo sembra molto infantile», scrive Skrenta, ma «ahimè! Non son più stato capace di liberarmi del mio flagello elettronico. Ha contagiato tutti i miei dischi e i dischi dei miei amici. È riuscito addirittura ad arrivare ai dischi con i programmi per i grafici di funzione del mio professore di matematica.» Skrenta ha inventato un programma per distruggere il virus, ma non si è mai dimostrato efficace quanto il virus stesso.

Quanto detto implica un problema interessante, e sarei privo di fantasia e irresponsabile se non lo ponessi: descrivere, in una pagina o meno, DOTTOR DOS, un programma su disco che in qualche modo sappia soffocare epidemie elettroniche di questo genere. Molti dischi usati da un calcolatore personale contengono copie del suo DOS. Quando viene fatto partire, il calcolatore ottiene dal disco la sua copia del DOS, che rimarrà operante anche quando vengono fatti girare altri dischi, sia pure contenenti anch'essi copie del DOS. Se è infetto, il DOS attivo può alterare le altre copie del DOS o addirittura sostituirle con copie di se stesso. Come ci si può opporre a questa virulenza?

Nella versione iniziale della Guerra dei nuclei, la difficoltà principale consisteva nel fornire al programma di battaglia A i mezzi per proteggersi dai colpi vaganti del programma di battaglia B. Se questo tipo di protezione riusciva a essere più o meno efficace, l'evoluzione del gioco portava al livello successivo, in cui i programmi sarebbero stati costretti a cercarsi l'un l'altro e a sviluppare attacchi concentrati.

Nel tentativo di garantire questa protezione, nell'articolo precedente avevo suggerito l'istruzione

PCT *A*

dove *A* è l'indirizzo relativo (diretto o indiretto) di un'istruzione che deve essere protetta. Un tentativo di cambiare il contenuto di quell'indirizzo sarebbe bloccato da MARS, il sistema supervisore del gioco. Il tentativo successivo, però, avrebbe avuto successo. Mi sembrava che, impiegando un semplice ciclo, un programma di battaglia potesse proteggere tutte le proprie istruzioni da bombe vaganti abbastanza a lungo da riuscire a lanciare una sonda indisturbata verso l'altro programma. La figura di questa pagina mostra in forma schematica un programma ad autoprotezione di questo genere. Il ciclo di protezione è formato da sei istruzioni, quattro delle quali eseguite a ogni passaggio nel ciclo. Così, un programma di battaglia di n istruzioni (ciclo incluso) richiederebbe circa $4 \times n$ esecuzioni per avere una protezione completa da un singolo colpo. Questo scudo non è una gran protezione contro un programma DWARF che lanci due colpi contro ogni locazione.

Esiste un altro uso di questa istruzione, non previsto nell'articolo precedente sulla Guerra dei nuclei. Stephen Peters di Timaru, Nuova Zelanda, e Mark A. Durham di Winston Salem, North Carolina, l'uno indipendentemente dall'altro, hanno pensato di usare PCT in modo offensivo. Un programma chiamato TRAP-DWARF innalza uno sbarra-

DAT		0	/puntatore all'indirizzo da proteggere
ADD	#1	—1	/incrementa l'indirizzo della protezione
PCT	@—2		/protegge l'indirizzo
CMP	#102	—3	/se tutte e 102 le istruzioni sono protette...
JMP	2		/... allora lascia il ciclo
JMP	—4		altrimenti riprendilo

⋮

CORPO DEL PROGRAMMA DI BATTAGLIA PRINCIPALE

⋮

Questo ciclo protegge i combattenti da bombe vaganti

mento di zeri nel solito modo ma poi protegge ogni deposito contro la sovrascrittura. Questo significa che un programma nemico incauto può cadere in una di queste trappole mentre si trascrive in una nuova area. L'istruzione indirizzata alla locazione occupata da uno zero protetto non avrebbe ovviamente effetto su quella locazione. Quando, in seguito, l'esecuzione raggiunge quell'indirizzo, il nuovo programma muore perché 0 non è un'istruzione eseguibile. Varrà forse la pena di includere PCT in qualche futura versione della Guerra dei nuclei, ma per ora la terrò da parte per amore di semplicità, sigillo del progettista del gioco.

Tra le altre idee suggerite dai lettori ci sono la matrice di nuclei bidimensionale proposta da Robert Norton di Madison, Wisconsin, e la regola di limitazione di campo avanzata da William J. Mitchell del dipartimento di matematica della Pennsylvania State University. L'idea di Norton si spiega da sé; la proposta di Mitchell, invece, richiede un approfondimento. Essa consiste nel consentire a ogni programma di battaglia di modificare il contenuto di qualsiasi locazione che non disti più di un certo numero prestabilito di indirizzi. Una regola di questo genere impedisce automaticamente a DWARF di fare danni al di fuori di questo intorno. La regola ha anche molti altri effetti, tra cui quello di sottolineare notevolmente il movimento: in quale altro modo un programma di battaglia potrebbe raggiungere il campo di un avversario? La regola ha molti pregi e spero che qualcuno dei molti lettori che possiedono un sistema per la Guerra dei nuclei le voglia dedicare l'ulteriore approfondimento che merita.

Norton propone anche che, in una battaglia della Guerra dei nuclei, a ogni contendente sia consentita più di una esecuzione. La stessa idea è venuta a molti altri lettori e ho deciso di accettare il suggerimento, così che ora la Guerra dei nuclei assume un carattere di grande apertura che prima mancava.

Il gioco si modifica aggiungendo la seguente istruzione, chiamata scissione, al listato ufficiale della Guerra dei nuclei (*si veda l'illustrazione a pagina 31*).

SPL *A*

Quando l'esecuzione raggiunge questo punto, si divide in due parti, l'istruzione che segue SPL e quella distante *A* indirizzi. Questo consente immediatamente a ogni giocatore della Guerra dei nuclei di far girare più programmi alla volta, quindi è necessario definire il modo in cui MARS assegnerà queste esecuzioni. Sotto questo profilo, esistono due diverse possibilità.

Per illustrarle, supponiamo che un giocatore abbia i programmi A_1, A_2 e A_3, mentre l'altro giocatore ha i programmi B_1 e B_2. Un'alternativa è far girare tutti i programmi del primo giocatore, seguiti da quelli del secondo giocatore. L'ordine dell'esecuzione sarebbe così A_1, A_2, A_3 e poi B_1 e B_2, e il ciclo si ripeterebbe indefinitamente. La seconda possibilità è alternare i programmi dei due giocatori. In questo caso la successione sarebbe A_1, B_1, A_2, B_2, A_3, B_1 e così via. I due schemi sono molto diversi. Il primo mette l'accento su una proliferazione illimitata e sembra quindi limitare il ruolo dell'intelligenza nel gioco. Nel secondo, invece, quanto maggiore è il numero di programmi fatti girare dai due giocatori, tanto minore è il numero di volte che ciascuno di essi sarà eseguito. In questo contesto sembra appropriata una legge dei ritorni decrescenti, quindi ho adottato il secondo schema. Scopo del gioco, in ogni caso, è provocare l'arresto di tutti i programmi nemici.

La nuova istruzione è ricca di possibilità creative. Per illustrarne una delle più modeste, ecco un programma di battaglia chiamato IMP GUN:

```
SPL 2
JMP -1
MOV 0 1
```

Consideriamo quello che avviene quando l'esecuzione arriva per la prima volta alla sommità di questo programma. SPL 2 significa che in seguito saranno assegnate a questo programma due esecuzioni: saranno eseguite sia JMP −1 sia MOV 0 1. La prima istruzione farà sì che il programma rientri nel ciclo e la seconda mette in movimento un IMP. L'IMP si muoverà verso il basso, naturalmente, dato che l'obiettivo del comando MOV sarà sempre l'indirizzo successivo, come indicato dall'1 (positivo). L'IMP così viene generato a ogni ciclo del programma e un flusso senza fine di esecuzioni di IMP scorre attraverso il nucleo deciso a distruggere i programmi nemici.

A prima vista, può sembrare che non vi sia alcuna difesa possibile contro un simile esercito di IMP; in realtà una esiste.

È necessario mettere in gioco IMP PIT, un programma ancora più semplice, che viene attivato da un comando SPL inserito in un insieme più esteso di istruzioni volte a proteggere il suo fianco superiore:

```
MOV #0 -1
JMP -1
```

A ogni esecuzione, IMP PIT pone uno zero subito sopra di sé, nella speranza di distruggere un IMP in arrivo. Qui è fondamentale la regola di esecuzione-assegnazione. Se IMP GUN appartiene ad A e IMP PIT appartiene a B, allora A richiede *n* mosse per eseguire *n* IMP; solo un IMP può arrivare alla locazione subito sopra l'IMP PIT. A parità di altre condizioni, B deve eseguire IMP PIT solo una volta per eliminare un IMP in arrivo.

Nella versione allargata del gioco della Guerra dei nuclei, si immagina che ogni contendente generi e metta in campo piccoli eserciti di programmi formulati singolarmente per individuare, attaccare, proteggere e anche riparare. Numerose sottigliezze, come quella proposta da John McLean di Washington, D.C., richiedono un'analisi ulteriore. McLean immagina un programma trappola specializzato, che sistema comandi JMP in vari indirizzi in tutta la matrice del nucleo nella speranza di far approdare un comando JMP all'interno di un programma nemico. Ogni JMP collocato in questo modo trasferirebbe l'esecuzione del programma nemico al programma trappola, provocandone per così dire il passaggio al nemico.

Dalla mischia provocata dai programmi di battaglia emerge un problema importante, che ha bisogno di soluzione. Che cosa impedisce a un programma di battaglia di uno dei contendenti di attaccare i suoi colleghi? Appare necessario un sistema di ricognizione.

Tra i molti lettori che hanno costruito sistemi per la Guerra dei nuclei meritano una citazione particolare: Chan Godfrey di Wilton, Connecticut, Graeme R. McRae di Monmouth Junction, New Jersey, e Mike Rosing di Littleton, Colorado, perché hanno messo particolare cura nel definire e documentare i loro progetti. Mi piacerebbe in particolare rendere disponibili ai lettori i documenti di Rosing, ma ho un'altra idea migliore che include questa possibilità e risolve anche altri problemi di comunicazione. Se qualche lettore con un sistema per la Guerra dei nuclei già funzionante si offrirà come direttore di una rete di Guerra dei nuclei, allora si potranno comunicare a tutti gli utenti della Guerra dei nuclei una documentazione dei vari sistemi, proposte di regole, programmi interessanti e battaglie. Un volontario sarà scelto come direttore; gli altri volontari potrebbero dar vita, secondo i loro interessi, a un bollettino, a un comitato per le regole, e così via.

Il programma MICE si fa strada fino alla vittoria nel primo torneo di Guerra dei nuclei

di A. K. Dewdney

Le Scienze, marzo 1987

La Guerra dei nuclei - il gioco in cui programmi specializzati fanno del loro meglio per distruggersi l'un l'altro - è stata alla ribalta del primo torneo internazionale che ha preso lo stesso nome, tenutosi al Computer Museum di Boston, Massachusetts. Fra i 31 programmi ammessi, tre si sono distinti per la loro forza; la vittoria finale è andata a un programma chiamato MICE. Il suo autore, Chip Wendell di Rochester, New York, ha ricevuto un bellissimo trofeo in cui era inserita una scheda di memoria a nuclei di ferrite di un vecchio calcolatore CDC 6600.

Anche se vengono creati da esseri umani, i programmi per la Guerra dei nuclei sono del tutto autosufficienti quando combattono nell'arena della memoria di un calcolatore. La sezione di memoria riservata alla lotta è detta nucleo, dal nome di un tipo ormai obsoleto di memoria costituita da anelli ferromagnetici miniaturizzati chiamati nuclei. Il gioco ha suscitato un entusiasmo tale da portare alla nascita della Società internazionale delle Guerre dei nuclei (International Core Wars Society). Questa società ha di recente modificato il gioco, alla cui nuova versione si atterranno d'ora in poi i giocatori.

La Guerra dei nuclei ha alla base un programma da battaglia - di cui erano equipaggiati anche i contendenti del recente torneo - scritto in un particolare linguaggio di basso livello denominato Redcode. Un insieme di 10 semplici istruzioni permette a un programma di spostare informazioni da una locazione di memoria a un'altra, di aggiungere o togliere informazioni, di modificare l'ordine in cui vengono eseguite le istruzioni e perfino di provocare la contemporanea esecuzione di più istruzioni (*si veda l'illustrazione a pagina 37*). Un'istruzione fondamentale, per esempio, è il comando di spostamento MOV, formato da tre parti - un codice di istruzione e due indirizzi - che occupano la stessa locazione nel nucleo. Il comando è scritto nella maggior parte dei casi sotto la forma MOV *A B*. Se *A*, poniamo, è 102 e *B* è −5, il calcolatore andrà all'indirizzo 102 e ricopierà ciò che c'è lì nella locazione che si trova cinque indirizzi indietro rispetto all'istruzione MOV.

Il più semplice dei programmi scritti in Redcode è formato da una sola istruzione MOV: MOV 0 1. Il programma, chiamato IMP, provoca il trasferimento del contenuto dell'indirizzo relativo 0 (vale a dire la stessa istruzione MOV) all'indirizzo relativo 1, esattamente un indirizzo avanti a se stesso. Le istruzioni in Redcode di solito sono eseguite una di seguito all'altra. Questo significa che, dopo l'esecuzione dell'istruzione MOV 0 1, il calcolatore cercherà di eseguire una istruzione all'indirizzo successivo. Naturalmente ora c'è un'istruzione a occupare quell'indirizzo: il comando MOV 0 1 appena copiatovi. Di conseguenza, IMP avanza di indirizzo in indirizzo per il nucleo, in modo ciecamente distruttivo, lasciando dietro in sé una scia di istruzioni MOV 0 1.

Un IMP può anche rubare a un programma nemico quella che è la sua vera anima, l'esecuzione. Per capire come possa accadere, immaginiamo che un programma da battaglia sia in esecuzione al solito modo, secondo l'ordine delle sue istruzioni. Un IMP si introduce nel programma dall'alto, sovrapponendo al codice un'interminabile successione di istruzioni MOV 0 1. Presto o tardi, il programma sovvertito ritrasferirà probabilmente l'esecuzione alla sezione invasa. A quel punto il programma diventa un nuovo IMP. Batte sempre la stessa bandiera ma ora è costretto a seguire le tracce dell'IMP nemico fino alla conclusione della battaglia.

Per evitare di essere invaso, un programma per la Guerra dei nuclei deve contenere come minimo un IMP-STOMPER, una protezione formata da due istruzioni eseguite ciclicamente:

```
MOV # 0 −1
JMP −1
```

Il primo comando sposta l'intero 0, simbolizzato da # 0, all'indirizzo relativo −1; in altre parole, ogni volta che viene eseguito il comando MOV si riempie con uno 0 la locazione immediatamente precedente (l'unica direzione da cui può arrivare un attacco di IMP). La seconda istruzione è il comando JMP che, quando viene eseguito, trasferisce il flusso di esecuzione, o di controllo, all'indirizzo nella locazione relativa −1, cioè l'indirizzo immediatamente precedente JMP. Ogni ciclo di esecuzione del programma fa sì che venga piazzato uno 0, così cancellandolo, su un qualsiasi IMP eventualmente arrivato subito sopra l'IMP-STOMPER.

Ci sono due regole fondamentali nella Guerra dei nuclei. La prima regola è che i due programmi in competizione devono alternarsi nell'eseguire le loro istruzioni e questa alternanza è gestita da MARS, il *Memory Array Redcode Simulator* (simulatore di Redcode nella matrice di memoria). Come suggerisce la sigla un po' militaresca, MARS simula l'azione di un calcolatore aggiornando di continuo il contenuto della matrice del nucleo secondo le istruzioni in corso di esecuzione. Compiendo questa operazione, permette a ogni contendente di eseguire una sola istruzione per volta. La seconda regola afferma che se un programma smette di girare ha perso.

Mentre gira, un programma può avere più di un flusso di esecuzione. Se l'esecuzione incontra il comando SPL *A* in un programma in Redcode, si scinde in due flussi, uno dei quali va all'istruzione che segue immediatamente SPL *A* e l'altro salta all'istruzione dell'indirizzo relativo *A*. Sfortunatamente, il sistema MARS non può eseguire simultaneamente entrambe le istruzioni; ne esegue una al turno successivo e l'altra al turno ancora dopo. Viene così un po' ridimensionato quello che si potrebbe ritenere un incredibile vantaggio: quanti più sono i flussi di esecuzione concomitanti di un programma, tanto più lentamente procede ciascun flusso. Del resto è giusto che sia così. Nel caso di flussi di esecuzione multipli, un programma di battaglia viene dichiarato vincitore quando tutti i flussi del suo avversario sono «morti». A questo punto MARS, che si aspetta ancora un'istruzione eseguibile, può trovare solo l'equivalente computazionale dei crateri di granate.

Per illustrare il comando SPL, riporto qui di seguito le prime cinque istruzioni del mio programma al torneo di Guerra dei nuclei. Si chiama COMMANDO per ragioni che risulteranno presto chiare.

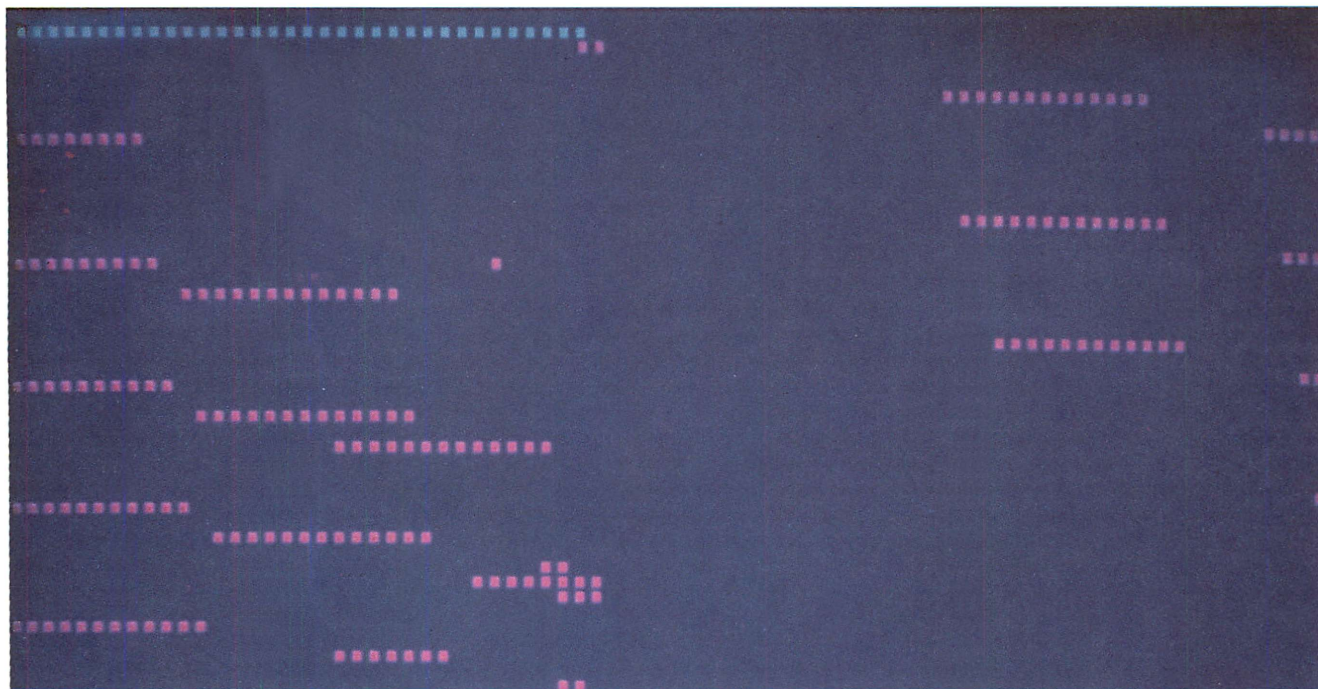
```
MOV # 0 −1
JMP −1
SPL −2
MOV 10 113
SPL 112
  .
  .
  .
```

I lettori riconosceranno nelle prime due istruzioni un IMP-STOMPER. L'esecuzione del programma vero e proprio inizia alla terza istruzione, SPL −2. Nei due turni successivi, COMMANDO eseguirà la

prima e la quarta istruzione e nei due turni dopo di questi eseguirà la seconda e la quinta istruzione. Ogni flusso procede in modo indipendente dall'altro e, per così dire, a mezza velocità. Nel codice appena esposto, COMMANDO lascia funzionare per conto suo l'IMP-STOMPER. Poi sposta un altro IMP (in paziente attesa 10 indirizzi al di là della seconda istruzione MOV) in una lontana locazione (113 indirizzi al di là). Il secondo IMP è attivato dal secondo comando SPL.

Le altre istruzioni di COMMANDO copiano l'intero programma in un nuovo segmento del nucleo, 100 indirizzi al di là della sua locazione attuale. La nuova copia, come un commando appena paracadutato in territorio nemico, è attivata da un comando JMP del programma originale. La vecchia copia di COMMANDO, a esclusione dell'IMP-STOMPER, cessa di funzionare; quindi inizia nuovamente l'intero ciclo di copiatura.

Come si è comportato COMMANDO contro i suoi avversari? Il torneo era stato organizzato in modo da consentire il maggior numero di scontri tra i 31 concorrenti. Un girone all'italiana completo, in cui ciascun programma combattesse contro tutti gli altri, avrebbe richiesto 465 incontri: più di quanto il tempo consentisse. Di conseguenza, i concorrenti sono stati ripartiti in modo arbitrario in due gruppi quasi uguali, la I e la II divisione; per ciascun gruppo si è svolto un girone all'italiana.

Potete immaginare lo strano miscuglio di emozioni che ho provato quando COMMANDO è risultato vincitore della II divisione. Da una parte ero fiero che il mio figlio cibernetico si fosse comportato così bene. Allo stesso tempo ero un

Stadio iniziale e finale di una battaglia tra MICE (rosso) e CHANG1 (azzurro)

po' imbarazzato dall'eventualità di vincere definitivamente il torneo. Dato che avevo acconsentito a fungere da commentatore per le finali, la mia obiettività (e credibilità) ne sarebbe stata indubbiamente inficiata.

I primi quattro programmi di ciascuna divisione sono stati ammessi a un nuovo girone all'italiana, da cui sono usciti vittoriosi tre programmi, CHANG1 di Morrison J. Chang di Floral Park, New York, e due concorrenti di Chip Wendell, MIDGET e MICE. Il mio COMMANDO ha dovuto arrendersi, mortalmente ferito. La vittoria finale di MICE è giunta in modo strano; sia MIDGET che MICE hanno terminato in pareggio la lotta contro CHANG1, ma MICE ha guadagnato il punto decisivo battendo MIDGET.

Lo scontro tra ogni coppia di finalisti consisteva in quattro battaglie consecutive. Il tempo limite per ciascun assalto era di 15 000 istruzioni per concorrente, ossia circa due minuti di tempo reale. I due programmi di battaglia prendevano il via da due posizioni distinte, scelte a caso. Tutte le battaglie tra una data coppia di programmi hanno avuto sempre lo stesso risultato. Nel caso di MICE contro CHANG1, si sono avuti quattro risultati di parità.

È affascinante osservare lo sviluppo di una Guerra dei nuclei. Il visualizzatore utilizzato nel torneo mostra il nucleo come una successione di strisce cellulari (*si veda l'illustrazione della pagina precedente*). Ciascuna cellula rappresenta un singolo indirizzo del nucleo e l'ultima cellula della riga in basso è contigua alla prima cellula della riga in alto, in accordo con la struttura circolare del nucleo. Il programma che muove per primo occupa inizialmente l'indirizzo 0 e in seguito riempie locazioni consecutive del nucleo. Il suo colore è l'azzurro. L'avversario occupa un segmento, scelto a caso, di locazioni non sovrapposte a quelle assegnate al primo programma. Il colore del secondo programma è rosso. Il colore di una cellula sul video dipende da quale è stato l'ultimo programma a modificarne l'indirizzo. Così si ha una visione avvincente dell'azione.

Contro un cielo blu scuro, MICE e CHANG1 iniziarono ad avanzare, lanciandosi IMP, sganciando bombe e riproducendosi (per partenogenesi). Uno degli scontri ebbe un andamento tipico: CHANG1 iniziò sotto forma di striscia di cellule azzurre nell'angolo in alto a sinistra dello schermo e la nascita di MICE fu annunciata dalla comparsa di una striscia rossa all'incirca a metà dello schermo. Immediatamente, MICE cominciò a proliferare con rapidità.

MICE, uno dei programmi ad autoriproduzione più brevi che io conosca, ha appena otto istruzioni, due delle quali creano una nuova copia del programma a circa 833 indirizzi di distanza dalla sua attuale locazione nel nucleo (*si veda l'illustrazione a pagina 38*).

Le due istruzioni evidenziano altre caratteristiche del linguaggio Redcode:

ciclo MOV @ *ptr* <5
DJN *ciclo ptr*

La parola *ciclo*, un'etichetta che sta semplicemente per un indirizzo, rende più facili da scrivere i programmi per la Guerra dei nuclei. Il comando DJN (abbreviazione di *decrement and jump on nonzero values*, ossia «diminuzione e salto per valori diversi da zero») fa saltare l'esecuzione all'istruzione etichettata *ciclo* se il valore immagazzinato in un altro indirizzo (contrassegnato con *ptr*) è diverso da zero. Il segno @ indica un sistema di riferimento noto come indirizzamento indiretto; quando viene eseguito il comando MOV, non sposta il contenuto della locazione contrassegnata con *ptr*, bensì sposta, per così dire, il contenuto del contenuto. Il numero immagazzinato in *ptr* è l'indirizzo del dato da spostare, in questo caso una delle istruzioni di MICE.

Il numero immagazzinato in *ptr* cambia in continuazione a causa della funzione decrescente del comando DJN. Il numero inizia all'ultimo indirizzo del programma e decresce progressivamente fino a zero; a quel punto il ciclo di copiatura è finito. In modo analogo, anche l'indirizzo in cui vanno immagazzinate le istruzioni è dato dall'indirizzamento indiretto. L'indirizzo relativo 5 contiene inizialmente il numero 833 e la prima istruzione spostata da MICE atterra 832 indirizzi al di là del comando MOV; come è indicato dal segno ‹, l'indirizzo bersaglio è diminuito e MOV viene eseguito. MICE crea una copia di se stesso a partire dalla coda.

Segue subito dopo un comando SPL, che trasferisce l'esecuzione alla nuova copia di MICE. Dopo questa nascita riuscita, però, il programma genitore inizia di nuovo. Non c'è limite al numero di figli che un programma di questo genere può produrre, e ogni nuovo programma fa la stessa cosa.

Fu così che, in un tipico scontro con CHANG1, MICE si riprodusse con incredibile rapidità e ben presto lo schermo si riempì di piccole strisce rosse. Nel frattempo, CHANG1 aveva attivato una specie di fabbrica di IMP all'estremità inferiore del suo flusso, fabbrica realizzata con tre sole istruzioni:

SPL 2
JMP −1
MOV 0 1

Quando l'esecuzione arriva al comando SPL, si divide in due rami. Uno trasferisce l'esecuzione a MOV 0 1, l'altro esegue l'istruzione JMP −1, che inizia daccapo il processo. Nel frattempo, un IMP è già partito in missione lasciando la catena di montaggio. Un problema legato alla sfrenata produzione di IMP è che un gran numero di flussi indipendenti rallenta l'esecuzione di ciascun processo: 1000 IMP si spostano 1000 volte più lentamente e faticosamente di un unico IMP. A ogni modo, l'orda fatale si materializzò nella parte alta dello schermo sotto forma di una solida striscia azzurra in continua crescita. Sarebbe riuscita a sconfiggere MICE?

ISTRUZIONE	MNEMONICA	ARGOMENTI		SPIEGAZIONE
Enunciato di dati	DAT		B	Enunciato non eseguibile; *B* è il valore dei dati
Sposta	MOV	A	B	Sposta il contenuto dell'indirizzo *A* all'indirizzo *B*
Somma	ADD	A	B	Somma i contenuti dell'indirizzo *A* e dell'indirizzo *B*
Sottrai	SUB	A	B	Sottrae il contenuto dell'indirizzo *A* dall'indirizzo *B*
Salta	JMP	A		Trasferisce il controllo all'indirizzo *A*
Salta se zero	JMZ	A	B	Trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* è zero
Salta se diverso da zero	JMN	A	B	Trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* è diverso da zero
Decrementa: salta se diverso da zero	DJN	A	B	Sottrae 1 dal contenuto dell'indirizzo *B* e trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* è diverso da zero
Confronta	CMP	A	B	Confronta i contenuti degli indirizzi *A* e *B*; se sono uguali, salta l'istruzione successiva
Dividi	SPL	A		Divide l'esecuzione nell'istruzione successiva e nell'istruzione in *A*

Riepilogo di Redcode, un linguaggio di assemblatore per la Guerra dei nuclei

CHANG1				MICE			
	MOV	#0	–1	*ptr*	DAT	#0	
	JMP	–1		*start*	MOV	#12	*ptr*
	DAT		+9	*ciclo*	MOV	@*ptr*	<5
start	SPL	–2			DJN	*ciclo*	*ptr*
	SPL	4			SPL	@3	
	ADD	#–16	–3		ADD	#653	2
	MOV	#0	@–4		JMZ	–5	–6
	JMP	–4			DAT	833	
	SPL	2					
	JMP	–1					
	MOV	0	1				

I partecipanti al campionato di Guerra dei nuclei

Mentre gli IMP si riproducevano, alcune delle copie di MICE venivano uccise da bombe di dati di CHANG1. Una bomba di dati consiste in genere in uno 0 lanciato da un comando MOV in quello che si spera essere il territorio nemico. L'istruzione chiave del programma di Chang è MOV# 0 @–4. Lo 0 è spostato su un indirizzo contenuto in una locazione che si trova quattro istruzioni sopra il comando MOV. La locazione è continuamente aumentata di 16 per garantire uno sbarramento ben intervallato.

Mentre qualche MICE moriva in questo modo, gli IMP iniziarono a esercitare la loro influenza distruttiva. Ogni copia del programma MICE originale, però, porta con sé un'opzione suicida; verifica continuamente se la sua prima istruzione (un enunciato di dati formato solo da 0) sia ancora zero. Se così non è, MICE fa avanzare l'esecuzione a un enunciato di dati (non eseguibile) e muore silenziosamente piuttosto di perdere l'anima a opera di un diavoletto.

Se alcune copie di MICE venivano uccise da bombe di dati e altre procedevano, per così dire, alla propria esecuzione per evitare la cattura, come fece MICE a sopravvivere? La risposta sta sicuramente nella sua sfrenata procreazione di nuove copie. Molte di queste, dopo tutto, atterravano su IMP nemici. In effetti, prima del termine una copia di MICE era riuscita ad atterrare sul programma natale di CHANG1 e lo aveva distrutto. CHANG1, però, aveva creato abbastanza IMP da riuscire a tener duro fino al fischio finale. La battaglia era pari.

L'arte della programmazione per la Guerra dei nuclei è ancora sicuramente ai suoi primi passi, ma il progresso sarà continuo. Alcuni intrepidi programmatori scopriranno infallibili rimedi contro gli IMP e altri scopriranno semplici strumenti di automanutenzione. I lettori che vogliano tenersi aggiornati sugli sviluppi possono abbonarsi a «The Core War Newsletter» scrivendo a William R. Buckley, 5712 Kern Drive, Huntington Beach, California 92649. I lettori che vogliano scrivere programmi di battaglia dovrebbero iscriversi alla International Core Wars Society. Attuale presidente è Mark Clarkson, che volentieri accetterà nuovi membri. Il suo indirizzo è 8619 Wassall Street, Wichita, Kansas 67210-1934. Non è però necessario iscriversi per ordinare a Clarkson l'importantissimo documento intitolato «Core War Standards» in cui si descrivono sintassi e semantica dei programmi in Redcode; il suo costo è di quattro dollari e senza questo documento non si può diventare veri combattenti dei nuclei.

I programmi di battaglia del futuro saranno forse più lunghi dei programmi che hanno partecipato al primo torneo internazionale di Guerra dei nuclei, ma saranno di gran lunga più protetti e potenti. Guadagneranno certamente in intelligenza, lasceranno tracce false e colpiranno i loro avversari improvvisamente e con determinazione. Queste tendenze risulteranno forse già evidenti nel secondo torneo internazionale di Guerra dei nuclei che è in programma al Computer Museum. Nel frattempo, i lettori hanno a disposizione ampie possibilità per mettere in luce la loro intelligenza e abilità nel linguaggio Redcode.

Il torneo dell'autunno 1986 deve parte del successo ottenuto a Mark e Beth Clarkson, a Gwen Bell, direttore del Computer Museum, e a Oliver Strimpel, suo direttore associato e curatore. Mi sembra opportuno concludere con una breve nota sul museo stesso.

Il Computer Museum di Boston è l'unico museo al mondo interamente dedicato ai calcolatori. Ospitato in un rinnovato (e ora elegante) deposito del porto di Boston, dispone di vecchi mostri a valvole, di calcolatori personali per giochi individuali, di un sistema completo NORAD SAGE e di un gran numero di oggetti idonei a divertire e a istruire. I lettori in visita a una famosa nave antica ancorata nel porto di Boston possono prendere una tazza di tè computazionale proprio alla porta accanto.

Un sistema esperto batte i semplici mortali nella conquista dei Sotterranei del Destino

di A. K. Dewdney

Le Scienze, aprile 1985

Ogni anno, migliaia di persone incontrano una morte di fantasia nei Sotterranei del Destino giocando ad Avventuriero (Rogue, in inglese), un rappresentante di una nuova generazione di giochi d'avventura con il calcolatore. Il giocatore osserva una mappa dei sotterranei sullo schermo e dirige le azioni di un personaggio chiamato l'Avventuriero. Obiettivo del gioco è scendere attraverso i 26 livelli dei sotterranei, impadronirsi dell'Amuleto di Yendor e tornare sani e salvi in superficie raccogliendo quanto più oro possibile lungo il percorso e uccidendo o sfuggendo ai mostri che si incontrano. Sono ben pochi i giocatori umani che riescono, non dico a tornare con l'amuleto, ma anche solo a sopravvivere ai pericoli di questa odissea sotterranea.

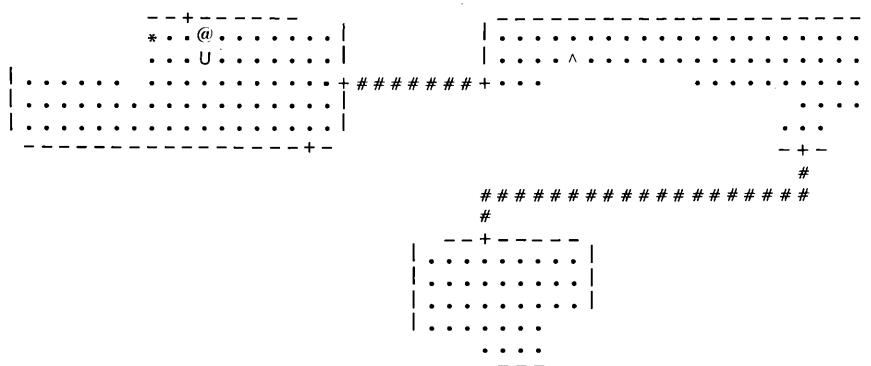
La sensazione di entrare davvero in un mondo fantastico può essere molto vivida; è facile lasciarsi rapidamente alle spalle la coscienza di essere davanti a una tastiera e a uno schermo. Come l'Avventuriero, mi avvicinai all'ingresso dei sotterranei con una certa trepidazione: era notte e le antiche rovine che indicavano il punto della mia imminente discesa avevano un aspetto lugubre e minaccioso. Preparai la mia mazza incantata, un arco e una faretra di frecce sottratti dal tesoro di un drago sulle Montagne Oscure. Indossai la mia armatura costruita dai folletti, afferrai le mie armi e i viveri e mi inoltrai nell'oscurità stigea di una scala.

Proprio quando la discesa cominciava ad apparirmi senza fine, mi trovai di fronte a una porta di quercia e la aprii con cautela. Davanti a me c'era la prima stanza del livello superiore dei Sotterranei del Destino. Delle candele spandevano una fievole luce e io mi spinsi fino al centro della camera, per sorvegliarla meglio. Improvvisamente, il pavimento cedette sotto i miei piedi e per un terribile secondo mi trovai a precipitare, finché con un tonfo doloroso atterrai in un'altra stanza. Questa era molto più buia e, anche dopo essermi abituato all'oscurità, non riuscivo a vedere che per pochi passi intorno a me. Per peggiorare la situazione, sentivo qualcosa muoversi nella stanza. Attanagliato dalla paura, inciampai e mi trovai di fronte a un'orrenda figura accovacciata, coperta da un'armatura e con in pugno una clava. Appena lo vidi sollevare la sua arma per colpirmi, un flusso di adrenalina mi schiarì le idee: con un unico fluido movimento misi una freccia sul mio arco, tesi la corda e scoccai la freccia. (Per mia fortuna, a scuola avevo seguito un corso di tiro con l'arco.) Con un sibilo e un urlo, il demonio (perché di un demonio si trattava) cadde a terra fremendo di rabbia. Con cautela uscii di lì, deciso a trovare una scala per tornarmene fuori dai Sotterranei del Destino. Pensavo alla mia casa accogliente e alla scrivania con l'articolo delle «(Ri)creazioni al calcolatore» da finire.

Mentre procedevo lentamente verso il punto in cui pensavo dovesse trovarsi una scala, colpii con lo stivale qualcosa che sembrava un mucchietto di pietre. Abbassai lo sguardo. Anche al buio qualcosa splendeva. Oro! Quasi quasi potevo esplorare un'altra stanza, una sola, prima di tornare in superficie...

A parte l'intensità dell'esperienza di immedesimazione, Avventuriero va al di là dei tradizionali giochi d'avventura sotto almeno due punti di vista. In primo luogo, la configurazione del terreno è generata dal programma stesso e varia da una partita all'altra. In secondo luogo, Avventuriero fornisce al giocatore una vista in pianta del tratto, esplorato fino a quel punto, del livello dei sotterranei in cui il giocatore si trova (*si veda l'illustrazione di questa pagina*). Varie caratteristiche sono indicate da diversi caratteri della tastiera che appaiono sullo schermo. Per esempio, trattini e sbarrette rappresentano pareti, i segni «diesis» e i segni di percentuale rappresentano rispettivamente passaggi e scale, i più rappresentano entrate e i segni ∧ rappresentano trappole. Oggetti e caratteristiche della camera possono essere o non essere visibili a seconda che la camera sia illuminata o meno. Quando l'Avventuriero penetra in questo livello, le caratteristiche del livello non appaiono sullo schermo; perché vengano rappresentate, l'Avventuriero (rappresentato da un segno @) deve scoprirle. Così egli esplora, incappa in qualche trappola, perlustra parti di varie stanze e attraversa passaggi. Può anche incontrare la lettera *U*, che rappresenta un mostro chiamato il Cupo Hulk.

Sotto la mappa grafica, lo schermo mostra l'attuale situazione statistica dell'Avventuriero. Nella figura, l'Avventuriero si trova al venticinquesimo livello, ha accumulato 7730 pezzi d'oro, ha guadagnato 77 punti (ma gliene rimangono solo 25 a causa dei danni sofferti), ha accumulato forza fino al quindicesimo grado (18 è il massimo), indossa una armatura di classe 9 e ha 30 668 punti d'esperienza, sufficienti per porlo all'elevato tredicesimo ordine d'esperienza. Al venticinquesimo livello dei Sotterranei del Destino, all'Avventuriero non rimane che una scala da scendere per raggiungere il livello inferiore e cercare di impadronirsi dell'ambito Amuleto di Yendor. L'Avventuriero, però, deve prima sbarazzarsi del Cupo Hulk.

In questa sede possiamo analizzare solo superficialmente il gioco. Sarebbe possibile scrivere un intero libro di regole e consigli per giocare ad Avventuriero. Per ora, l'unico manuale del genere disponibile è un conciso opuscolo di otto

Il venticinquesimo livello dei Sotterranei del Destino

@	l'Avventuriero	h	spostati di uno spazio a ovest
–|	pareti di stanze	j	spostati di uno spazio a sud
+	vano di porta	k	spostati di uno spazio a nord
#	passaggio tra stanze	l	spostati di uno spazio a est
·	pavimento di stanza	>	discendi la scala
%	scala	<	sali la scala
∧	trappola	m	spostati su qualcosa
)	arma	s	ricerca se ci sono trappole
]	pezzo di armatura	∧	identifica la trappola
*	oro	. /e	riposa/mangia
!	fiaschetta di pozione	W/T	indossa/togli l'armatura
?	pergamena magica	P/R	metti/togli l'anello
:	cibo	d	abbandona un oggetto
/	bacchetta magica	q	bevi una pozione
=	anello magico	r	leggi una pergamena
A–Z	le lettere maiuscole	z	elimina con la bacchetta magica
	indicano abitanti	t	getta un oggetto
	dei sotterranei	w	impugna un'arma
		f	combatti all'ultimo sangue

Simboli della mappa* (a sinistra) *e comandi per il gioco Avventuriero

pagine scritte da Michael C. Toy e Kenneth C. R. C. Arnold, *A Guide to the Dungeons of Doom*.

Sfortunatamente è disponibile soltanto il manuale per gli utenti del sistema a partizione di tempo VAX/UNIX. A quanto mi dice Arnold, è ora disponibile una versione di Avventuriero adatta per l'IBM PC e la si può richiedere (sotto la denominazione inglese di Rogue) alla A. I. Design, 201 San Antonio Circle, Suite 115, Mountain View, California 94040. Toy e Arnold, i creatori di Avventuriero, pensano evidentemente che il modo migliore per imparare il gioco sia giocarlo. È comunque possibile dare una descrizione a grandi linee delle caratteristiche generali del gioco.

A ogni livello, l'area di gioco è divisa in quadrati. L'Avventuriero, mentre esplora l'ambiente che lo circonda, occupa un quadrato alla volta. Il movimento all'interno delle stanze e dei passaggi è controllato mediante comandi formulati premendo alla tastiera lettere quali *h, j, k* e *l* che muovono l'Avventuriero di un unico quadrato in una delle quattro direzioni principali. Altri comandi producono un movimento diagonale o un movimento continuo in una data direzione. Per far salire o scendere una scala all'Avventuriero, il giocatore deve battere il carattere < o, rispettivamente, >.

Quando si scopre un oggetto vicino alla posizione dell'Avventuriero, per raccoglierlo automaticamente bisogna spostare l'Avventuriero sul quadrato in cui si trova l'oggetto. Se il giocatore vuole spostarsi su quel quadrato senza raccogliere l'oggetto, deve battere una *m* seguita dal carattere per la direzione appropriata. Battendo la *s*, si può anche far effettuare all'Avventuriero una ricerca delle trappole nei quadratini circostanti; con l'avvertenza, però, che la ricerca ha solo il 20 per cento di probabilità di portare effettivamente alla scoperta di una trappola.

Di tanto in tanto, l'Avventuriero deve fermarsi (battere un puntino) o mangiare (battere una *e*) per recuperare le forze spese nell'esplorazione o nella lotta contro i mostri. A parte una limitata quantità di cibo trasportata in uno zaino, l'Avventuriero non ha altro da mangiare che quello che può trovare per terra nei sotterranei (per quanto poco invitante possa sembrare).

Quando trova un pezzo di armatura, l'Avventuriero può prenderselo (mettendolo automaticamente nello zaino) o indossarlo (battere una *W*). Ovviamente, l'armatura dà all'Avventuriero maggiore protezione in combattimento, ma può anche essere stregata. Se l'armatura è stregata, ogni tentativo di togliersela (battere una *T*) è inutile senza una pergamena magica che rompe la maledizione. Tutti gli anelli magici trovati dall'Avventuriero possono essere indossati (battere una *P*) o tolti (battere una *R*) a meno che non siano stregati.

A volte l'Avventuriero è costretto ad abbandonare una fiaschetta e una pergamena o due (battere una *d* seguita da un carattere che rappresenta l'oggetto) perché lo zaino è pieno. (Un Avventuriero con lo zaino pieno non può raccogliere un'armatura.) Prima di abbandonare la fiaschetta, però, l'Avventuriero può berne l'ignoto contenuto sperando di trarne beneficio: alcune pozioni hanno un effetto curativo e altre mettono in grado l'Avventuriero di vedere il Mostro Invisibile. D'altro canto, la pozione può provocare uno stato di disorientamento; può capitare, ad esempio, che dopo aver dato all'Avventuriero il comando di muoversi verso nord, egli cominci a vagare in una direzione a caso. Allo stesso modo, può essere vantaggioso leggere una pergamena (battere una *r*) prima di gettarla: certe formule tolgono la maledizione da un'armatura.

Gli strumenti magici più impressionanti a disposizione dell'Avventuriero sono delle bacchette. A seconda del tipo di bacchetta raccolta, l'Avventuriero può sbarazzarsi di un mostro (battere una *z*) con vari effetti: può trasportarlo in un luogo a caso, lanciargli addosso globi infuocati o trasformarlo in un altro mostro. È meglio usare quest'ultimo tipo di bacchetta, detta bacchetta polimorfa, sui mostri più orribili: è di gran lunga preferibile trasformare il terribile Verme Purpureo in un pipistrello che viceversa.

Quando l'Avventuriero scopre un mostro, può essere opportuno lasciarlo stare: a volte un mostro sta dormendo e non attaccherà se lasciato in pace. Se però un combattimento appare inevitabile, si può impugnare un'arma (battere una *w*) e lottare all'ultimo sangue (battere una *f*), dopo aver spostato l'Avventuriero vicino al mostro. L'esito dello scontro dipenderà in modo probabilistico da fattori quali l'attuale livello di forza dell'Avventuriero, il suo grado di esperienza e la classe dell'armatura.

Tra i giocatori di Avventuriero, primeggia un giocatore non umano di grosso calibro: un programma per calcolatore che nel corso degli ultimi quattro anni si è dedicato a questo gioco rivaleggiando in prodezza con i migliori concorrenti alla ricerca dell'amuleto e dell'oro. Questo programma fornisce anche un'interessante opportunità di osservare un sistema esperto al lavoro.

Il 16 febbraio del 1984, all'Università del Texas di Austin, un Avventuriero controllato da un programma fece fuori tutti i mostri, ammassò un considerevole mucchio d'oro e ritornò con l'amuleto. Questo programma, dal nome ben poco magico di ROG-O-MATIC, diresse ogni passo dell'Avventuriero, ogni sua pausa, ogni lancio e ogni colpo.

ROG-O-MATIC è la creazione di quattro studenti del dipartimento di scienze del calcolatore della Carnegie-Mellon University di Pittsburgh: Andrew Appel, Leonard Hamey, Guy Jacobson e Michaël Mauldin. ROG-O-MATIC unisce fonti di conoscenza programmate a sistemi esperti per prendere decisioni sul da farsi in ogni immaginabile situazione sotterranea.

Quando un essere umano gioca ad Avventuriero, un flusso di comandi passa dalla tastiera al programma ROGUE grazie al sistema operativo. Il programma decide automaticamente quale mostro schierare contro l'Avventuriero, che configurazione usare per il prossimo livello dei sotterranei, e così via. Il programma trasmette queste informazioni, sempre attraverso il sistema operativo, allo schermo per tenere informato l'essere umano della situazione attuale.

Sostituendo un giocatore umano, il programma ROG-O-MATIC intercetta il flusso di caratteri da tastiera e trasmette per proprio conto caratteri al programma ROGUE. Quest'ultimo non ha nessuna idea (per così dire) del fatto che sta giocando un altro programma. Allo stes-

so modo, anche l'informazione diretta dal programma ROGUE verso lo schermo viene diretta verso il programma ROG-O-MATIC, che può così avere la propria mappa dei Sotterranei del Destino.

Il programma ROG-O-MATIC è formato da 12 000 righe scritte nel linguaggio di programmazione C ed è ancora più lungo e complicato del programma ROGUE. ROG-O-MATIC iniziò nel 1981 con qualcosa che i suoi primi creatori, Appel e Jacobson, pensavano dovesse essere un «progetto semplice». Poco dopo che Mauldin si fu unito al gruppo, il primo programma che giocava ad Avventuriero passò attraverso una larga serie di modifiche dando luogo a diverse varianti, ciascuna delle quali aggiungeva qualche miglioramento nella tattica o nella strategia. Quando anche il quarto membro, Hamey, iniziò a contribuire allo sviluppo del programma, gli autori cominciavano a rendersi conto di aver creato, in effetti, un sistema esperto. Questi sistemi costituiscono un'applicazione chiave della cosiddetta Quinta Generazione di calcolatori, che secondo i progetti dovrebbe dare risultati commerciali alla fine degli anni ottanta. Un sistema esperto è progettato per incorporare e proiettare l'abilità umana in una grande varietà di aree, dalla medicina all'ingegneria.

Usando il tipo di architettura software adatta per sistemi esperti, i creatori di ROG-O-MATIC furono in grado di delineare e modificare il loro programma con relativa facilità. In particolare, organizzarono i vari tipi di conoscenza e abilità richiesti dall'Avventuriero in una gerarchia di sottosistemi diversi (*si veda l'illustrazione in questa pagina*).

Per esempio, un esperto di alto livello (chiamato Mischia) controlla la lotta durante il combattimento e un altro esperto di alto livello (chiamato Bersaglio) dirige la caccia dell'Avventuriero ai mostri. Entrambi questi esperti utilizzano un esperto di livello inferiore, detto *battaglia,* che effettua particolari attacchi oppure inizia una ritirata a seconda della situazione. L'esperto in battaglia chiama a volte in aiuto l'esperto in ritirata e quest'ultimo attinge invariabilmente a una fonte di conoscenza chiamata *perc,* un particolare algoritmo che studia il terreno alla ricerca del più breve percorso possibile per qualsiasi posizione specificata. In generale, gli algoritmi per il percorso più breve hanno avuto un notevole sviluppo e in questa particolare applicazione è accettabile solo l'algoritmo più veloce possibile: viene utilizzato quasi in continuazione nel corso dell'esplorazione dei Sotterranei del Destino da parte di ROG-O-MATIC. Le conoscenze sul terreno usate da *perc* vengono da *termap,* una struttura di dati che registra le caratteristiche del terreno scoperte fino a quel momento dall'Avventuriero nel corso della sua esplorazione di un particolare livello dei sotterranei. Infine, *termap* ricava tutte le sue conoscenze da *senso,* una struttura di dati di basso livello che contiene tutte le informazioni rilevanti fornite in uscita dal programma ROGUE.

Prima di elencare i compiti di altri esperti e di altre fonti di conoscenza, vorrei riesaminare più in dettaglio uno degli esperti, *battaglia.* Una volta che una battaglia è in corso, l'esperto in mischia richiede all'esperto in battaglia di decidere se attaccare o ritirarsi. Dato che il meglio del valore sta nella prudenza, l'esperto in battaglia stabilisce per prima cosa l'auspicabilità e fattibilità di una ritirata. Per farlo deve verificare l'esistenza di alcune condizioni preliminari:

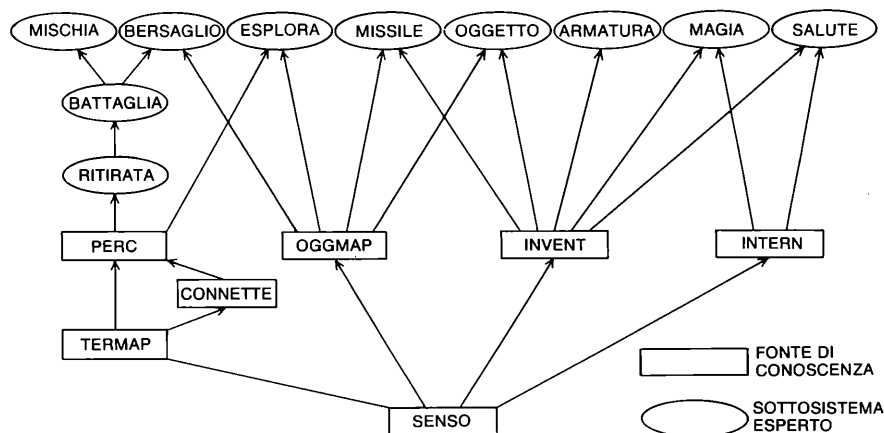
1. L'Avventuriero non è attualmente sotto l'influenza della pozione che produce confusione (che potrebbe far partire direttamente l'Avventuriero verso il mostro).
2. L'Avventuriero non è già tenuto saldamente da un mostro.
3. Sarebbe possibile morire nel corso di una mischia (in questo caso è fortemente auspicabile evitare il conflitto).
4. L'esperto in ritirata può trovare una via di fuga (una ritirata è possibile).

Se valgono tutte e quattro le condizioni, ROG-O-MATIC affiderà l'incarico della ritirata dell'Avventuriero all'esperto in ritirata. Se invece non valgono, l'esperto in battaglia passa in rassegna una serie di possibilità aggressive.

1. Se è possibile che l'Avventuriero muoia nel corso di una mischia, se il mostro è vicino e visibile e se l'Avventuriero si trova ad avere una bacchetta per il teletrasporto, allora punta la bacchetta contro il mostro.
2. Se è possibile che l'Avventuriero muoia nel corso di una mischia, se il mostro è vicino e se l'Avventuriero ha una pergamena per il teletrasporto, allora legge la formula magica riportata sulla pergamena.

Se non vale né l'una né l'altra di queste condizioni, non rimane altra alternativa che andare almeno una volta allo scontro con il mostro, quindi il programma ROG-O-MATIC impegna arditamente in battaglia il suo Avventuriero. Forse è abbastanza forte per sopravvivere a uno scontro; in caso contrario, può anche rimanere ucciso.

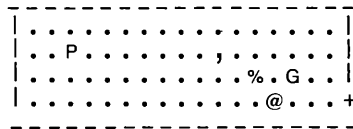
Tra gli altri esperti usati dal programma c'è un esperto in esplorazione *(esplora)* che decide dove condurre la prossima esplorazione e che movimento usare. Ci sono poi un esperto in missili, che controlla il lancio di frecce, massi, lance e così via contro i mostri, un esperto nella scelta degli oggetti da raccogliere, chiamato *oggetto,* un esperto nella scelta dell'armatura da indossare, un esperto nell'uso della magia e un esperto, chiamato *salute,* che decide quando mangiare o riposarsi. *Oggmap* è una struttura di dati che registra la posizione e la storia di tutti gli oggetti incontrati fino a un certo punto, *invent* è un inventario di ciò che è contenuto nello zaino dell'Avventuriero e *intern* è un riconoscitore di stato interno che controlla se l'Avventuriero è pronto a uno sforzo.

Non c'è da sorprendersi se ROG-O-MATIC può giocare rapidamente una partita di Avventuriero. Dopo pochi minuti di tempo del calcolatore è tutto finito in un modo o nell'altro. Fino a oggi, solo alla Carnegie-Mellon il programma ha giocato più di 12 000 partite di Avventuriero; i dati statistici riportati dai quattro creatori di ROG-O-MATIC tendono a confortare l'affermazione che attualmente il programma supera in abilità la grande maggioranza degli esperti umani di Avventuriero. Per esempio, in un test effettuato alla Carnegie-Mellon durante un periodo di tre settimane nel 1983, ROG-O-MATIC ottenne un punteggio medio superiore a quello dei 15 migliori giocatori di Avventuriero dell'università. ROG-O-MATIC supera la maggior parte degli esseri umani e mostra, rispetto a essi, alcune differenze notevoli nello stile di gioco. Secondo Mauldin, il programma è prudente e privo di immaginazione; esplora in modo efficiente ed evita tutte le ricerche ridondanti che gli esseri umani probabilmente intraprendono. Il suo stile di lotta, inve-

Sottosistemi esperti e fonti di conoscenza usati dal programma ROGUE

ce, è piuttosto metodico e, se non è fortunato, non è in grado di imitare le ardite possibilità del gioco umano.

Consideriamo, per esempio, il fattore fortuna nella sua storica impresa del febbraio 1984 all'Università del Texas di Austin. Dopo aver superato gli incredibili pericoli dei Sotterranei del Destino, l'Avventuriero (naturalmente sotto il controllo di ROG-O-MATIC) trovò l'Amuleto di Yendor in un passaggio del ventiseiesimo livello. Tornando verso la superficie con il suo trofeo, l'Avventuriero incontrò al ventiduesimo livello uno dei peggiori mostri, un drago che sputa fuoco. L'Avventuriero estrasse la spada ma il drago soffiò una saetta di fuoco per primo. La fiamma mancò l'Avventuriero, che in quel momento si trovava nel vano di una porta, colpì una parete e rimbalzò direttamente contro il drago, sbruciacchiandolo gravemente. Con le forze che gli rimanevano, il drago avanzò verso l'Avventuriero, che lo finì con un colpo di spada e poi proseguì verso la superficie occupandosi lungo la strada di altri mostri meno pericolosi. Quando emerse alla luce del giorno, aveva in suo possesso non solo l'amuleto d'oro ma anche 6913 pezzi d'oro e altri oggetti.

Il lettore che, incuriosito dal precedente resoconto, voglia provare il divertimento offerto dal gioco può impegnarsi nel seguente rompicapo costruito appositamente da Mauldin. Oltre a essere un'introduzione ai piaceri di Avventuriero, illustra la difficoltà di inserire intelligenza nel programma ROG-O-MATIC.

Una stanza del ventiseiesimo livello dei Sotterranei del Destino è attualmente occupata dal Verme Purpureo (*P*), dal Grifone (*G*) e dall'Avventuriero (@). L'Amuleto di Yendor (,) si trova sul lato opposto di una scala (%) rispetto all'Avventuriero.

L'obiettivo dell'Avventuriero è afferrare l'amuleto e correre sulla scala senza rimanere ucciso. All'Avventuriero rimane un solo punto da usare in lotta e non può rischiare uno scontro con il Verme Purpureo o con il Grifone. Per fortuna, il Verme Purpureo è profondamente addormentato e l'Avventuriero, indossando l'Anello della Segretezza, deve solo evitare di calpestarlo per non svegliarlo. Il Grifone, d'altra parte, è quanto mai sveglio e in caccia dell'Avventuriero. Il Grifone non disturberebbe mai un mostro amico, ma vorrebbe molto occupare la stessa posizione dell'Avventuriero, che smembrerebbe con un solo colpo dei suoi crudeli artigli.

Tocca all'Avventuriero muovere. Ogni mossa dell'Avventuriero o del Grifone porta all'occupazione di uno degli otto punti adiacenti. Il Grifone è così ristretto di vedute da scegliere sempre una mossa lungo il percorso più diretto verso la sua preda. Se l'Avventuriero si sposta sulla scala è al sicuro dal Grifone e dal Verme Purpureo: le regole di cooperazione tra i mostri proibiscono l'inseguimento su altri livelli.

Una tortuosa odissea da Robotropoli alle porte elettroniche di Silicon Valley

di A. K. Dewdney

Le Scienze, settembre 1985

Come galassie che entrano in collisione, i giochi e l'istruzione con il calcolatore sembrano prossimi a una grande fusione. Se ne è già avuto un annuncio l'anno scorso con la comparsa di Odissea del robot (Robot Odyssey), un gioco al calcolatore che è anche un mezzo per imparare a progettare circuiti.

Sono stati prodotti molti programmi didattici col calcolatore, ma nessuno, per quanto ne so, ha la naturale e coinvolgente intensità di Odissea del robot. Molti giochi al calcolatore sono passati dal caos primitivo dei videogiochi da sala a una solida struttura razionale; nessuno di loro, però, ha il valore educativo di Odissea del robot.

Odissea del robot non lascia al giocatore altra scelta se non quella di progettare circuiti logici per fuggire da Robotropoli, un fantastico scenario urbano abitato solo da robot. La maggior parte di questi automi mostra un'aperta ostilità a una presenza umana. Quando il giocatore (che sullo schermo compare sotto forma di un piccolo omino) cerca di superare un labirinto o di recuperare un oggetto da una stanza, un robot di guardia afferra l'omino e senza troppi riguardi lo rimanda alla partenza. Tre robot risultano preziosi compagni nella fuga. I circuiti dei robot amici, che i robot di guardia non conoscono, permettono loro di aggirare gli automi che abitano Robotropoli, superando labirinti o recuperando oggetti come gettoni della metropolitana e cristalli di energia.

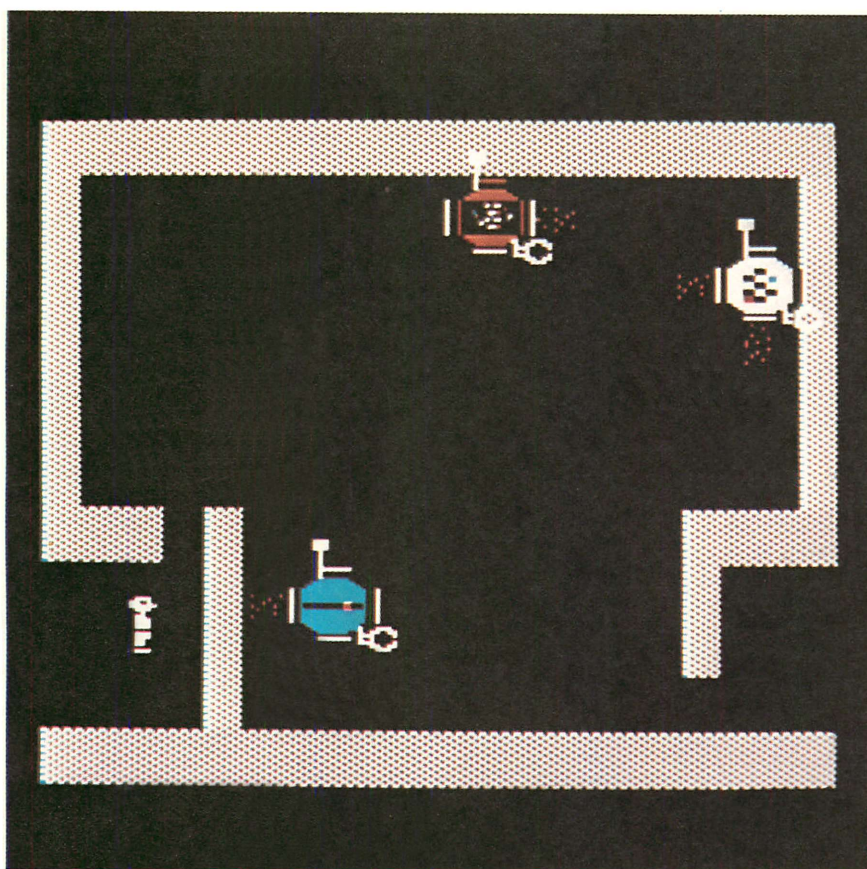
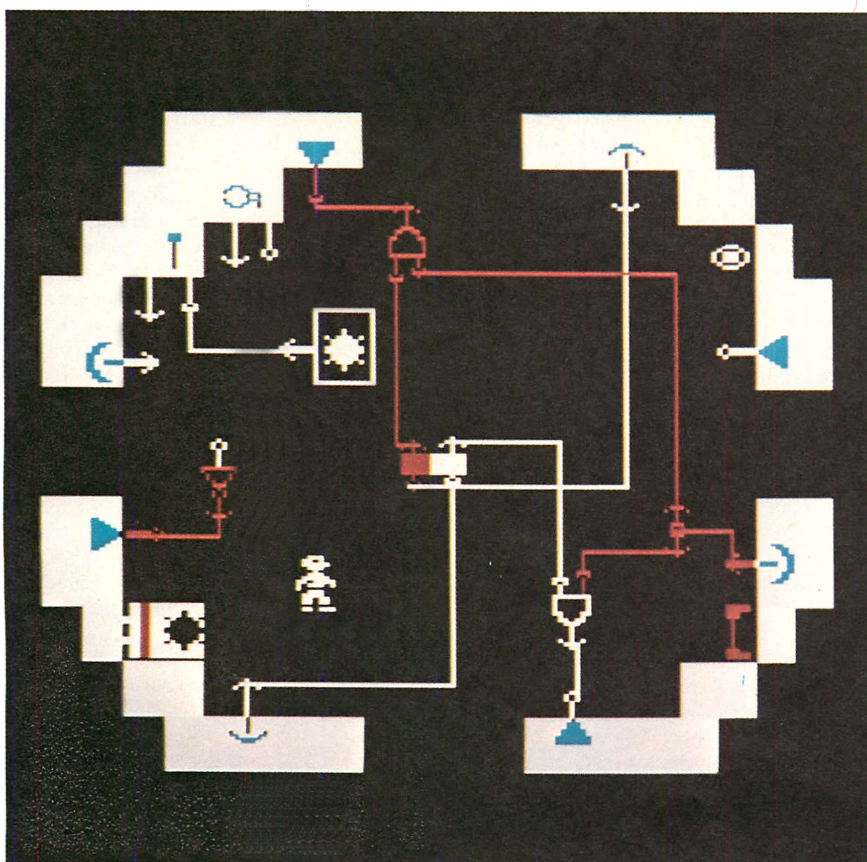
Il giocatore, dopo aver fatto un'odissea del robot, può progettare un calcolatore? Non proprio, ma l'esercizio è utile. Il gioco ha inizio in un luogo detto camera del robot. Qui l'omino, azionato dalla tastiera, incontra tre robot amici che rimbalzano allegramente in un'area rettangolare. Ogni robot si comporta in modo diverso: uno si muove per la stanza scivolando lungo le pareti, un altro rimbalza orizzontalmente avanti e indietro; il terzo va su e giù lungo la parete di destra come se vi stesse cercando un passaggio. Il giocatore inizia battendo la lettera *R* e la barra spaziatrice. Questa azione ha l'effetto surreale di trasformare l'omino in un'unità di controllo a distanza che comanda ai robot di fermarsi. A questo punto il giocatore può riportare l'unità di controllo remoto alla forma di omino. L'omino sale su uno qualsiasi dei robot e a questo punto allo scenario della stanza si sostituisce quello dell'interno cavo del robot. L'omino appare ridotto a un decimo delle sue dimensioni iniziali. Egli viaggia dentro il robot solamente quando deve superare un labirinto o in qualche altra situazione in cui può risultare utile la modalità di locomozione del robot.

Ciascun robot amico è attrezzato per percepire il proprio ambiente e spostarsi in esso. Dei respingenti gli segnalano il contatto con le pareti, meccanismi di spinta lo portano nelle quattro direzioni, una pinza può prendere o lasciare degli oggetti, un'antenna permette di comunicare con gli altri due robot amici. Nell'interno del robot tutto questo è rappresentato da connettori che possono essere cablati direttamente al circuito logico interno. I connettori sono di due tipi: connettori di ingresso, che comunicano al circuito lo stato di un sensore (respingente, pinza, antenna) e connettori di uscita, che collegano i comandi del circuito a un esecutore (meccanismi di spinta, pinza, antenna). La pinza e l'antenna, quindi, hanno connettori sia di ingresso sia di uscita. I connettori di uscita attivano la pinza e quelli di ingresso stabiliscono se nella pinza in quel momento c'è un oggetto.

Si può capire come funzionano i circuiti osservando operare il più semplice dei tre robot amici. Questo robot, chiamato Ping-Pong, viene controllato da un unico elemento flip-flop collegato a respingenti e meccanismi di spinta. In Odissea del robot, un flip-flop è una cella di memoria a due facce in cui, quando una faccia è attiva, l'altra è disattivata. Quando viene inserito, anche solo momentaneamente, l'ingresso della faccia disattivata, questa viene attivata mentre quella che era attiva viene disattivata. Se non si verifica alcun cambiamento ai suoi ingressi, il flip-flop mantiene il suo stato: in un certo senso, ricorda. Per fare agire Ping-Pong non è necessario alcunché di più complicato di un flip-flop: l'uscita di una faccia è collegata al meccanismo di spinta di destra del robot e l'ingresso della stessa faccia è collegato al respingente di destra. Se quella faccia del flip-flop è disattivata, quando il robot tocca una parete sulla destra l'ingresso del respingente viene attivato, il segnale viene portato al flip-flop e la faccia disattivata viene immediatamente attivata. Il meccanismo di spinta di destra entra in funzione e il robot comincia a muoversi a sinistra. E questo è il ping. Il pong viene predisposto da un collegamento analogo dell'altra faccia del flip-flop al respingente e al meccanismo di spinta di sinistra.

Per ottenere un comportamento più complicato è necessario l'intero insieme di elementi della logica standard. Questo insieme comprende non solo l'elemento di memoria flip-flop, ma anche elementi di decisione chiamati porte *or*, porte *and* e porte *not*, o invertitori. In poche parole, l'uscita di una porta *or* è attiva se almeno uno dei suoi ingressi è attivo mentre l'uscita di una porta *and* è attiva se entrambi i suoi ingressi sono attivi. L'uscita di un invertitore è attiva se il suo ingresso è disattivato.

Per andare oltre la camera dei robot, il giocatore di Odissea del robot deve recuperare una chiave che si trova all'interno di uno dei robot e aprire una porta che lo conduce in una regione che rappresenta la fogna della città. Per raccogliere un oggetto, bisogna sovrapporgli l'omino e premere la barra spaziatrice. Un bip annuncia il successo dell'operazione; dopodiché l'oggetto si muove insieme all'omino. Senza questa caratteristica Odissea del robot sarebbe impossibile: l'omino deve portare con sé i tre robot amici durante l'esplorazione dei cinque livelli del gioco. Ma come può trasportare tre robot se si può portare un solo oggetto alla volta? Il giocatore non ci metterà molto a pensare che, se un oggetto si può tenere all'interno o all'esterno del robot e se il robot stesso è un oggetto, può mettere un robot dentro all'altro. Funziona. In effetti l'omino può trascinare un robot che sta dentro un altro robot che si trova già all'interno di un terzo. Questa tattica permette di risparmiare spazio all'interno del terzo robot, che, col procedere dell'avventura, si riempie di gettoni della metropolitana, sensori, cristalli di energia e altri elementi del circuito.

La mia partita di Odissea del robot iniziò come un viaggio apparentemente senza fine attraverso una fila di camere adiacenti alla fogna della città. Attraversando la fogna giunsi a una stanza difesa con molto zelo da uno dei più abietti abitanti di Robotropoli. Nella parte posteriore della stanza, contro una parete, c'era un cristallo di energia. Lo desideravo ardentemente, così recuperai il robot che si muove lungo le pareti dall'interno di Ping-Pong e lo misi in moto. Dato che il robot scivolava lungo le pa-

L'entrata a Robotropoli e i tre robot che vi si trovano

Respingenti, meccanismi di spinta, antenne, connettori e circuiti del robot cercatore destro

reti della stanza, il guardiano non si accorse di lui. Il robot, quando trovò il cristallo di energia, lo raccolse nella pinza, emettendo un bip a indicare la riuscita dell'operazione e continuò a seguire le pareti dirigendosi verso l'entrata. Allora io disinserii il robot e cercai di prendere dalla pinza il cristallo di energia, ma il robot non lo voleva mollare. Rimandando a più tardi la soluzione del problema, rimisi il robot all'interno di Ping-Pong e mi preparai a continuare la mia odissea, quando si fece sentire un rumore stridente seguito dal piagnucoloso decrescendo che nel gioco segnala che la batteria è a secco. Un robot vampiro chiamato Ampir'Bot aveva conficcato la sua sonda elettrica in Ping-Pong e risucchiato metà della sua potenza. Si deve stare sempre all'erta a Robotropoli!

Dopo aver vagato raccogliendo qui e là oggetti che mi sembravano utili, giunsi a un passaggio che recava la scritta AVETE TUTTO? Non avevo tutto: avevo dimenticato di prendere un magnete in una stanza vicina alla fogna, perché temevo Ampir'Bot, che evidentemente teneva la zona sotto controllo. Il mio omino si aprì la strada oltre la scritta e giunse a un labirinto chiamato «grata della fogna» fatto di pareti verticali disposte a interrompere il percorso dal basso e dall'alto *(si veda l'illustrazione in basso della pagina seguente)* e custodito da un altro abietto robot.

Uno dei robot amici sembrava fatto apposta per contrastarlo. Quando il robot che chiamo cercatore destro è attivo, si muove sempre verso destra. Se una parete blocca il movimento verso destra, il robot scivola verso l'alto o verso il basso finché non va a urtare contro una parete orizzontale che gli fa immediatamente cambiare direzione. Le ragioni di questo comportamento risultano evidenti dando uno sguardo ai suoi collegamenti elettronici (*si veda l'illustrazione in basso di questa pagina*). Il meccanismo di spinta di sinistra è sempre attivo perché riceve un ingresso da un invertitore che non è collegato ad alcunché. (Dato che l'ingresso dell'invertitore è disattivato, la sua uscita è attiva.) Il controllo dei meccanismi di spinta verticali dipende da un elemento flip-flop connesso nello stesso modo del circuito che governa Ping-Pong. C'è, tuttavia, una semplice differenza: le uscite per i meccanismi di spinta verticali passano attraverso porte *and.* Il respingente di destra fornisce il secondo ingresso a ciascuna porta *and.* Questo significa che il comando per mettere in funzione o il meccanismo di spinta verso l'alto o quello verso il basso non sortisce alcun effetto finché il respingente di destra non tocca qualcosa.

Tirai fuori il robot cercatore destro dal robot che avevo trasportato e vi mandai sopra il mio omino. Sovrapponendo l'omino a un'icona a forma di occhio che si trova nel robot cercatore destro, ottenni una visione periscopica

dell'ambiente: l'interno del robot venne sostituito sul mio schermo dall'immagine del labirinto. Sulla sinistra c'era il cercatore destro con un periscopio alzato. Non mi era chiaro come la scena potesse comprendere il congegno visore, ma lasciai perdere la questione.

Finché rimaneva alzato il periscopio, non c'era nulla che impedisse al mio omino di diventare un'unità di controllo. Battei *R* e poi la barra spaziatrice. Il cercatore destro, sempre col mio omino, scivolò nella grata e urtò un muro. Andando in esplorazione verso l'alto, raggiungemmo la cima del labirinto e rimbalzammo verso il basso finché non trovammo un passaggio. Qui, al di sotto del labirinto, c'era una quantità di sfortunati omini che non erano riusciti a passare attraverso la grata. Evitammo il robot di guardia che sembrò non accorgersi della presenza umana che stava viaggiando sul cercatore destro. Attraversammo il passaggio, incontrammo un'altra parete verticale e procedemmo verso l'alto. Quando finalmente arrivammo dall'altra parte del labirinto mi resi conto di aver commesso un errore grossolano: avevo lasciato indietro gli altri due miei amici. La paura di Ampir'Bot mi aveva impedito di ragionare. Come potevo fare per tornare a prendere i due robot? Il cercatore destro doveva essere ricablato. Pensando un attimo ai suoi circuiti, i lettori capiranno probabilmente come feci.

Per eseguire il compito meccanico di ricablare un robot, si batte *S* e l'omino si trasforma in un saldatore. In questo modo la barra spaziatrice viene usata per rompere i vecchi collegamenti e per rifarne di nuovi. Se manca una parte, si batta *T* e comparirà sullo schermo un insieme di strumenti. L'omino vi può passeggiare in mezzo e scegliere l'elemento desiderato.

Quando ebbi finalmente radunato tutti e tre i robot dall'altra parte della grata della fogna, la mia odissea riprese. Per prima cosa raggiunsi una camera chiusa a chiave. Mi servii della chiave per entrare e scoprii che la stanza non aveva altre uscite. Dove si poteva andare di lì? Una parete recava un'inquietante iscrizione: PREPARATI PER UN'ESPERIENZA CHE TI TRASPORTERÀ IN UNA NUOVA DIMENSIONE. Nel mezzo della stanza uno spiritello, replica del mio omino, danzava all'interno di una scatola fissa.

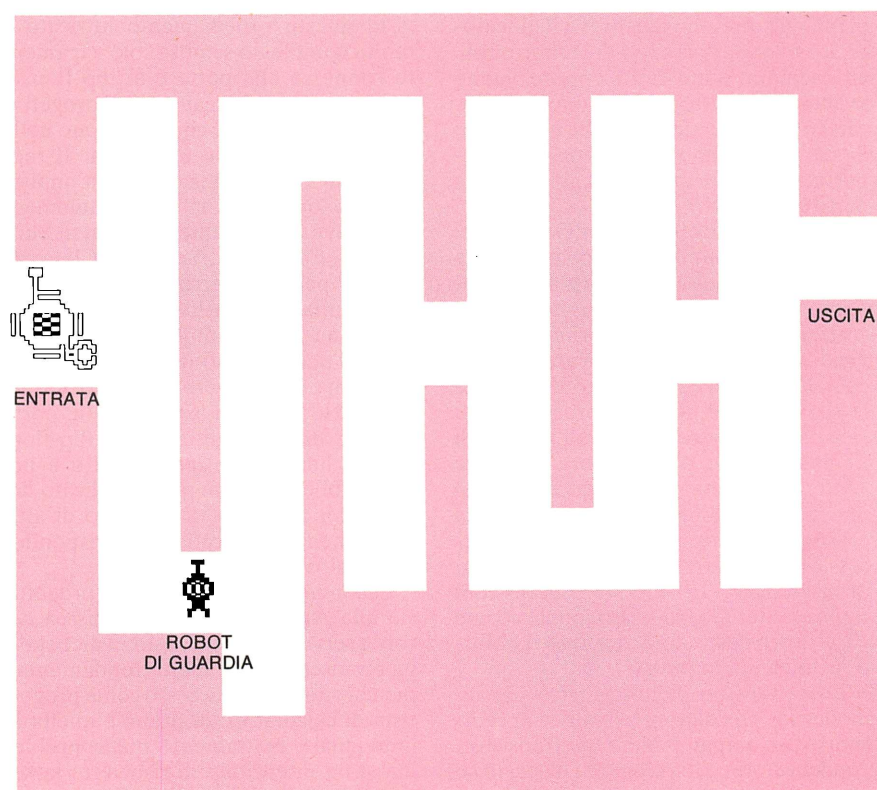
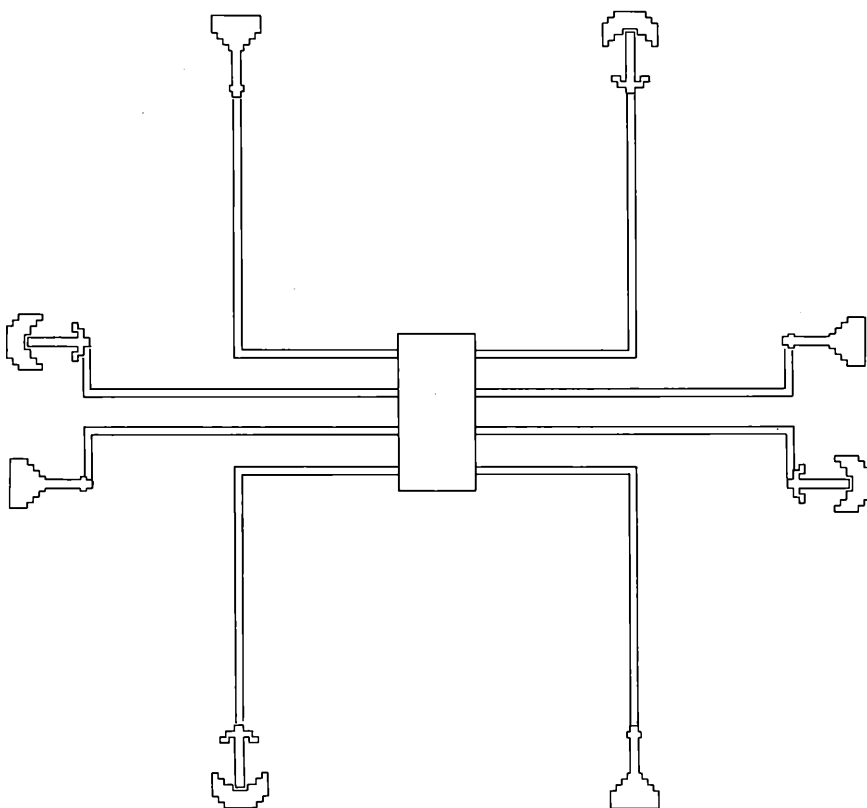
In Odissea del robot è possibile in qualsiasi momento salvare il gioco su disco. Se la mossa successiva risulta un disastro, il gioco può essere ripreso dalla posizione alla quale è stato salvato. Questa è una caratteristica di molti giochi di avventura col calcolatore. Si spera che i giovani patiti di questi giochi non crescano con l'impressione che anche quell'avventura che chiamiamo vita si possa salvare per poter giocare di nuovo.

Dopo aver salvato il gioco, sovrapposi il mio omino al suo duplicato e premetti la barra spaziatrice. Tutto sembrò esplodere in una doccia di scintille e uno strano suono accompagnò il mio spostamento in una nuova dimensione. Ero effettivamente atterrato in una nuova stanza al secondo livello di Robotropoli - senza i miei compagni. Chiaramente, si supponeva che dovessi portarmeli dietro. Servendomi del gioco che avevo salvato, mi fu possibile ripetere il trasferimento di dimensione, questa volta mantenendo una salda presa sui miei robot.

Descrivere le mie successive avventure in Robotropoli ridurrebbe il piacere dei lettori che volessero giocarlo di persona. Odissea del robot è venduto dalla Learning Company (545 Middlefield Road, Suite 170, Menlo Park, California 94025). Oltre al gioco, gli acquirenti ricevono dischi di apprendimento sull'anatomia del robot, l'uso dell'insieme di strumenti e la progettazione di chip. Quest'ultima caratteristica è racchiusa in un luogo detto laboratorio delle innovazioni. In qualsiasi momento del gioco un giocatore può visitare il laboratorio per verificare un nuovo progetto di circuito sui robot che vengono tenuti lì. Col crescere in grandezza e complessità dei circuiti è utile saperli trasferire su chip. Questo viene fatto nella camera di fusione del laboratorio.

Uno dei robot amici, quello che costeggia le pareti, era già attrezzato con un chip raffinato. Nella pagina successiva sono riportati in forma schematica i collegamenti del chip. I lettori possono divertirsi a progettare un circuito che costeggi le pareti da realizzare su un chip cosiffatto. Servitevi solo dei flip-flop e delle porte descritte. Sarò ben lieto di valutare i circuiti che mi invierete; alcuni lettori, però, desidereranno rendersi conto da soli di come funzionano i loro circuiti.

Il gioco Odissea del robot è stato progettato e scritto da Michael Wallace, quando era studente alla Stanford University, e da Leslie Grimm della Learning Company. Questo gioco ha avuto dei precursori, fra i prodotti della Learning Company. Sia in Rompicapo di Gertrude, sia in Gli stivali di Rocky è necessario che il giocatore progetti cir-

Elementi logici di Odissea del robot

La grata della fogna: un labirinto da risolvere con l'aiuto del robot cercatore destro

Respingenti, meccanismi di spinta e connettori del robot che costeggia le pareti. Che circuiti si troveranno all'interno del chip?

cuiti logici per risolvere problemi. Warren Robinett, già alla Learning Company, ha dotato tutti questi giochi di ambienti che mettono alla prova il giocatore. Le stanze e i passaggi di Robotropoli, ad esempio, sono finiti eppure vanno avanti all'infinito, almeno finché il giocatore non si rende conto che andarsene da destra da una stanza potrebbe comportare l'entrare in un'altra già visitata a partire da sinistra.

Progettare un circuito per Odissea del robot è un compito molto diverso da quello di progettarne uno per un moderno calcolatore. Tuttavia, è possibile costruire un calcolatore primitivo servendosi dei componenti forniti dal gioco. Chi diventa esperto può desiderare di scoprire se è possibile progettare un calcolatore che trovi posto all'interno di un unico robot. Il calcolatore potrebbe venir programmato fornendo in ingresso una successione di 0 e 1 proprio prima che il robot venga attivato. Questa successione equivarrebbe a un linguaggio ultrasemplice, le cui sottosuccessioni rappresenterebbero istruzioni come «Spostati a destra fino al muro» o «Metti in funzione la pinza».

Progettare un circuito logico per un moderno calcolatore significa predisporre un'ampia schiera di transistor, condensatori e altri elementi al silicio su un grande progetto immagazzinato in un calcolatore. Una volta verificato e corretto per simulazione, il progetto può essere trasferito su una pellicola fotografica e inciso su un chip di silicio. A prima vista questo procedimento intricato e molto complesso sembra solo vagamente connesso alle porte e ai flip-flop di Odissea del robot o ai corsi di progettazione logica di base che si tengono nella maggior parte delle università. Il rapporto mi venne spiegato alcuni anni fa da Christopher R. Clare, che attualmente si trova al CTX International di Sunnyvale, California. Secondo Clare, la ragione per introdurre alla progettazione circuitale attraverso la logica standard è quella di stimolare i futuri ingegneri al pensiero astratto. Di fronte a un problema da risolvere, il trucco sta nel trasferirlo al regno astratto delle porte *and, or, not,* e degli elementi flip-flop. L'abitudine al pensiero astratto è poi trasferibile al campo più complicato dei transistor a effetto di campo o di altri componenti sofisticati che corrispondono ad altri tipi di porte logiche.

Secondo Peter Ashkin, capo progettista alla Apple Computers, Odissea del robot serve allo scopo. Ashkin dice che il gioco insegna due abilità fondamentali per chi vuole aver successo come progettista di calcolatori: la prima è quella di individuare esattamente quale problema si ha intenzione di risolvere; la seconda è quella di formulare una soluzione in una logica, di quasiasi tipo. Per esempio, l'ultima volta che ho giocato a Odissea del robot mi sono trovato di fronte al problema di recuperare un gettone della metropolitana dal centro di una stanza custodita. Chiaramente, il problema era far sì che uno dei robot amici ricuperasse il gettone. Come avrei potuto raggiungere lo scopo? Il robot che si muove costeggiando le pareti non mi sarebbe stato di nessun aiuto perché non si sarebbe mai avvicinato abbastanza da prendere il gettone. Era disponibile un sensore di gettoni, fornito di uscite direzionali, che avrebbe certamente condotto il robot sul gettone; sfortunatamente, il congegno avrebbe anche impedito al robot di andarsene col suo trofeo. In ogni modo, analizzando il problema, si giungeva all'identificazione di due compiti distinti: progettare un circuito di ricerca del gettone e trovare un comportamento, di facile programmazione, che portasse il robot fuori della stanza. Ashkin osserva che, benché alcuni problemi siano molto nebulosi, si possono risolvere parti di essi di dimensioni ridotte. In tal modo il problema globale diventa progressivamente più definito. Ambedue i processi, definizione del problema e soluzione di parti di esso mediante la logica, appartengono al ciclo di progettazione.

Insegnando al giocatore come progettare circuiti, Odissea del robot offre anche un significativo vantaggio psicologico. Quando un giocatore acquista sicurezza nel cablare i robot amici, sviluppa una sensazione di padronanza del mezzo, padronanza che funziona da antidoto a quella sensazione di fredda lontananza che i calcolatori possono ispirare. Imparare la logica standard e tradurla in pratica nei robot di Odissea significa comprendere che tra le due cose esiste la stessa relazione che c'è tra gli elementi logici di base di un calcolatore e la sua funzione ultima. Si impara che non c'è alcun ostacolo a un'ulteriore indagine. Alla sensazione di mistero che emana dal calcolatore, si sostituisce quella di padronanza.

Secondo Sherry Turkle, autrice di *The Second Self,* la sensazione di padronanza dello strumento è una delle fasi che il neofita del calcolatore può attraversare. Il suo libro tratta dei rapporti che si sviluppano tra il calcolatore e i suoi utenti. Alcuni racconti hanno un'inquietante caratteristica: parlano di chi è giunto a considerare se stesso simile a una macchina. Sembra che tale opinione di sé sia possibile a tutti i livelli di abilità. So di scienziati e operatori nel campo dell'intelligenza artificiale che pensano di se stessi la stessa cosa, anche se a un livello più sofisticato. In un certo senso può essere vero che siamo tutti macchine, ma il saperlo ci dà poche indicazioni di comportamento e può incoraggiare l'adozione di modelli di ruolo disastrosi. Ecco perché, metaforicamente parlando, potrebbe essere utile acquisire l'abilità di fuggire da Robotropoli.

Star Trek emerge dalla clandestinità e trova il suo posto fra i videogiochi domestici

di A. K. Dewdney

Le Scienze, gennaio 1987

Nei primi anni settanta, mi capitava spesso di lavorare fino a tardi nel mio ufficio all'università, nella speranza di non essere interrotto dagli studenti. Sfortunatamente, proprio nel mio corridoio c'era anche il laboratorio di ricerca per la grafica al calcolatore e lì, ogni sera, si radunavano gli appassionati di un gioco allora in gran voga tra gli studenti e variamente denominato Star Trek o Guerra spaziale. «Prendilo! Prendilo!»; gli urli mi arrivavano attraverso la porta chiusa dell'ufficio. «Attento ai missili!» Il rumore era sufficiente a farmi perdere il filo dei miei pensieri. Incapace di battere Star Trek, in genere finivo con l'unirmi al gruppo dei giocatori: ripromettendomi di tornare a questioni meno frivole non appena ricaricata la mia attenzione con una pausa distensiva, facevo un giretto fino al laboratorio per osservare quello che stava avvenendo.

Il gioco Star Trek è liberamente ispirato all'omonima serie di telefilm; la competizione consiste in uno scontro tra l'astronave *Enterprise* e un incrociatore da battaglia di Klingon. Gli appassionati del programma televisivo ricorderanno che l'*Enterprise* viaggiava là dove gli esseri umani non si erano mai spinti e che l'esplorazione era condotta in nome di una coalizione di razze cooperanti detta Federazione, chiaramente governata dal popolo. I klingoniani erano gli irsuti rivali della Federazione nel progetto di dominazione sulla Via Lattea, se non sull'intero universo.

Se in origine Star Trek richiedeva un potente calcolatore per la ricerca grafica, ora può essere comodamente programmato su una macchina personale. Le due astronavi sono in orbita intorno a un sole centrale e lanciano missili una contro l'altra cercando di schivare il fuoco di risposta. Sia le astronavi sia i missili sono soggetti alla gravità e l'azione è in gran parte determinata dal moto orbitale. I piloti che non abbiano una discreta sensibilità per la meccanica celeste finiscono con il mandare la propria astronave a incenerirsi contro il sole oppure con l'incrociare senza avvedersene l'elusiva traiettoria orbitale di un missile. Perde chi salta in aria per primo.

Versioni di Star Trek erano apparse in centinaia di università e di altre istituzioni contemporaneamente; ufficialmente il gioco era guardato con un certo disprezzo, ma in segreto era molto stimato. È un'ironia che, a distanza di anni da quelle interruzioni serali, io ricordi quel gioco con affetto. Star Trek era solo uno dei tanti giochi sviluppati da studenti, giochi che in gran parte si affermarono come pacchetti commerciali contribuendo notevolmente alla rivoluzione dei calcolatori domestici. Di Star Trek si sono avute solo di recente versioni commerciali, anche se per qualche tempo sono state disponibili versioni per videogiochi da sala.

La versione di Star Trek descritta in questa sede riporta il lettore all'epoca clandestina e romanzesca dei primi giochi per calcolatore. Inoltre, serve per introdurre il complesso tema della programmazione di videogiochi. Uno dei compiti è quello di tener vivo lo schermo: i calcoli che creano l'azione devono essere quanto più veloci e semplici è possibile. Descriverò un problema analogo nell'articolo sui programmi di simulazione di volo a pagina 62. Fortunatamente, il mondo di Star Trek è molto più semplice della dettagliata geografia digitale che deve essere presentata ai piloti della tastiera. Un secondo obiettivo della programmazione è creare un ambiente gravitazionale realistico; in questo caso molti hanno incontrato lo stesso problema sotto forma un po' diversa. Nell'articolo a pagina 58 parlerò di un universo «da poltrona» in cui le stelle danzano seguendo la loro mutua attrazione gravitazionale. In Star Trek una gravità di tipo più semplice distorce le traiettorie di astronavi e missili. Soltanto il sole centrale esercita una forza percepibile.

Per ingaggiare il loro duello orbitale, due giocatori siedono alla tastiera di un calcolatore e premono i tasti che controllano le loro astronavi. Un giocatore, magari quello più peloso, prende il comando dell'astronave di Klingon; l'altro presiede alle fortune della Federazione. All'inizio lo schermo è occupato soltanto dal sole centrale e dalle due astronavi. Il sole è un cerchio e le astronavi sono icone con quel tanto di dettaglio sufficiente a distinguere l'amico dall'avversario.

Quando inizia il gioco, entrambe le astronavi sono in caduta libera verso il sole. Immediatamente i giocatori spostano le astronavi dalla linea di caduta e accendono i motori a razzo per portarsi in orbita di sicurezza. Se un'astronave tocca il sole si vaporizza all'istante, e naturalmente la partita è persa.

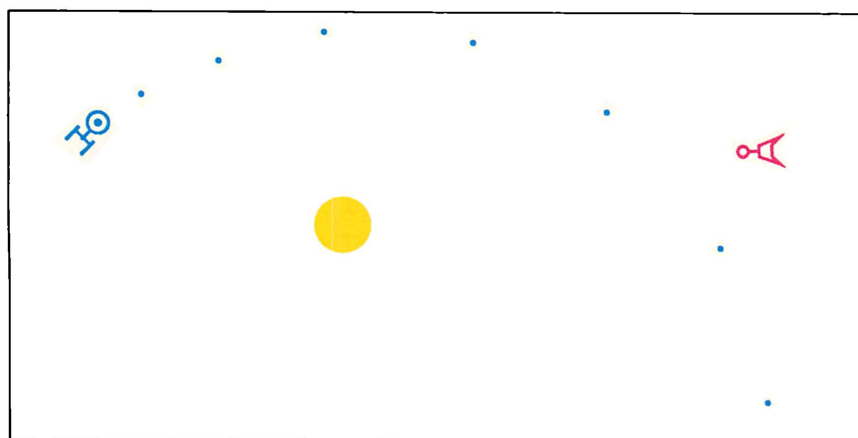
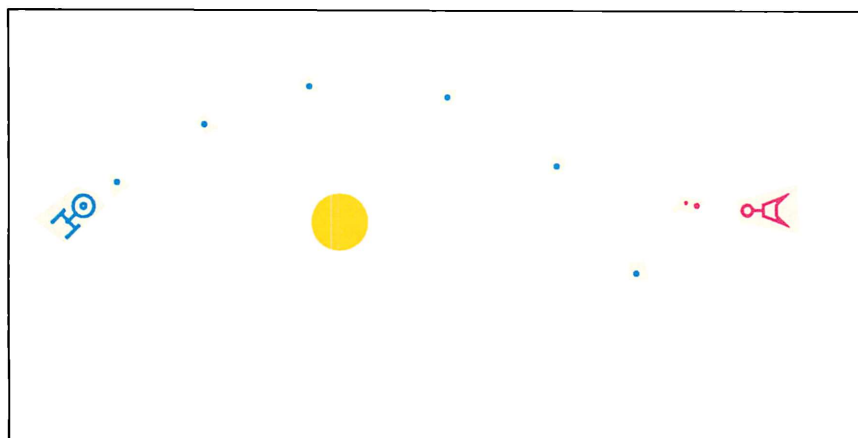
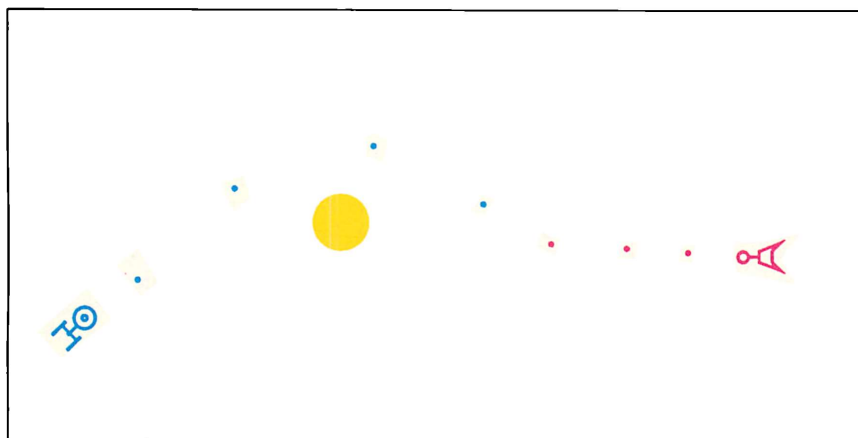
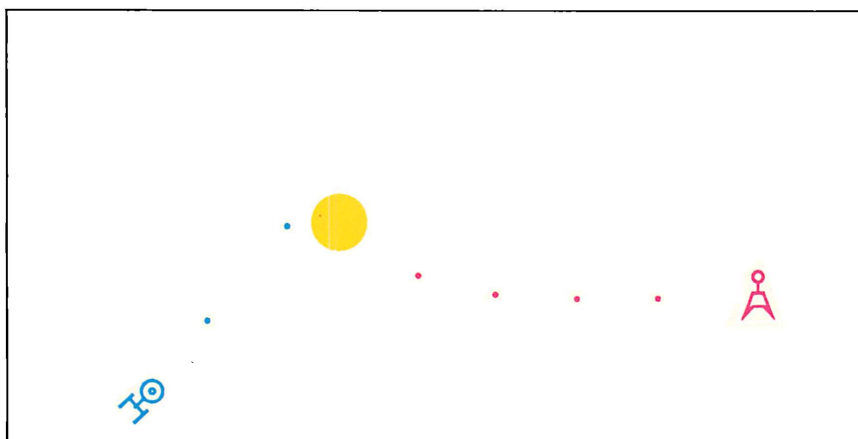
Appena stabilite le orbite, ciascun giocatore dà inizio ai tentativi di eliminazione dell'avversario. Una tattica estrema consiste nel rimanere in attesa fino al passaggio dell'astronave nemica nelle vicinanze e colpirla con una raffica veloce. A un altro estremo c'è una tattica più complicata: indirizzare i colpi da una posizione al lato opposto del sole rispetto all'astronave nemica. Sullo schermo appaiono brillanti punti luminosi - chiamati missili fotonici nel programma televisivo - che escono dal vascello spaziale che ha aperto il fuoco e che si muovono intorno al sole in un minaccioso schieramento leggermente incurvato. Se non è comandata da un pilota di straordinaria abilità, l'astronave nemica è destinata a incontrare uno dei missili e a esplodere in una miriade di detriti interstellari. Lo schermo segnala l'evento con una nuvola di puntini e annuncia VITTORIA PER LA FEDERAZIONE oppure VITTORIA PER LE FORZE DI KLINGON.

Data l'intensità dell'azione vicino al sole, i giocatori prudenti preferiscono scegliere orbite più esterne. Il principale svantaggio di questa strategia è la necessità di ricaricare una cella a energia solare. Il controllo di ogni astronave dipende dalla sua cella di energia; quando questa è esaurita, l'astronave deve avvicinarsi rapidamente al sole per fare rifornimento di fotoni solari. In un'orbita distante, il flusso di fotoni è debole e l'astronave corre il rischio di diventare un facile bersaglio.

C'è un'altra pericolosa caratteristica del gioco di cui bisogna tener conto nel volo lungo orbite esterne: lo spazio di battaglia in Star Trek è «toroidale». In altri termini, se l'astronave si avvicina troppo a un margine dello schermo, scomparirà e riapparirà vicino al margine opposto.

Ogni astronave ha un rifornimento infinito di missili, ma non ce ne possono mai essere più di 10 in volo nello stesso momento. I missili obbediscono alle stesse leggi a cui è soggetta l'astronave e un missile dura finché colpisce un'astronave (anche la propria) o finché rimane senza combustibile. Tanto basta per illustrare il gioco.

Il programma che io chiamo TREK è il più ambizioso tra quelli che presento in questo volume, non tanto per la sua complessità, quanto per la sua lunghezza. Di alcuni dei sottoprogrammi più comuni è possibile fare solamente un cenno.

I neofiti nel campo della programmazione, e in modo particolare coloro che si trovano già in collegamento con qual-

*Scambio di missili tra l'*Enterprise *e l'incrociatore di Klingon*

che istruttore attraverso la rete di addestramento, possono comunque provare ad affrontare il progetto con qualche speranza di successo e sicuramente con notevole divertimento.

Finché entrambe le astronavi sono operative, TREK si muove ciclicamente lungo sei sezioni principali di codice:

Leggere i tasti.
Aggiornare la posizione dell'astronave e dei missili.
Controllare gli eventuali contatti.
Aggiornare l'energia delle astronavi.
Gestire i missili.
Visualizzare.

Se si eccettua forse la prima sezione, molti lettori troveranno che TREK è relativamente semplice da scrivere. La lettura dei tasti risulterà nuova per qualcuno, ma questa caratteristica è indispensabile per la programmazione di videogiochi. La maggior parte dei linguaggi di alto livello ha istruzioni che permettono a un programma di controllare se è stato premuto un certo tasto.

Per esempio, nel linguaggio BASIC della Microsoft il comando in questione è

On key(k) gosub n.

Quando viene eseguito il comando «On key», il programma esegue un controllo all'inizio di ogni comando successivo per stabilire se è stato premuto o meno il tasto k. Se lo è stato, il programma passa al comando della riga n. Alla riga n inizia un sottoprogramma con il compito di registrare l'avvenuta pressione del tasto k, di solito assegnando un certo valore a una variabile che funge da indicatore. Ci sono 14 tasti specifici sulla tastiera del PC IBM che possono essere verificati in questo modo: i 10 tasti funzione e i quattro tasti per il controllo del cursore. Si devono assegnare quattro tasti a ogni giocatore - diciamo i quattro tasti funzione $F1$, $F2$, $F3$ e $F4$ alle forze della Federazione e i quattro tasti del cursore alle forze di Klingon. I numeri per questa assegnazione sono rispettivamente quelli da 1 a 4 e quelli da 11 a 14. A questo punto è indispensabile poter consultare un manuale.

Per ciascuno dei giocatori, il primo dei quattro tasti controlla la spinta, i due successivi controllano la direzione e l'ultimo controlla il lancio dei missili. Il tasto della spinta impartisce semplicemente una spinta prefissata, a piena potenza, per un ciclo di programma. Ogni pressione di un tasto di direzione fa girare l'astronave corrispondente di un angolo di 10 gradi. Un colpetto sul tasto dei missili provoca il lancio di un singolo missile. Prima di iniziare il gioco è opportuno incollare delle piccole etichette sui tasti di controllo, con simboli scelti in modo da ricordare ai combattenti la funzione di ciascun tasto.

Il programma TREK deve includere un comando «On key» per ciascuno degli

otto tasti di controllo designati. «On key» non viene eseguito a meno che sia preceduto dal comando «Key(k) on».

Tutti i sottoprogrammi richiamati dai comandi «On key» sono semplici e sostanzialmente uguali. Ciascun sottoprogramma è formato da due istruzioni. La prima pone uguale a 1 una variabile indicatore per una successiva consultazione da parte del programma; la seconda provoca un ritorno dell'esecuzione del programma al numero di riga del comando «On key» che aveva chiamato il sottoprogramma. Gli indicatori per il controllo dell'*Enterprise* potrebbero chiamarsi *fdav*, *fddt*, *fdst* e *fdfu*, che starebbero per «Federazione avanti» (accendere il propulsore), «Federazione a destra» (girare in senso orario), «Federazione a sinistra» (girare in senso antiorario) e «Federazione fuoco» (lanciare un missile). In modo analogo, le variabili *knav*, *kndt*, *knst*, *knfu* registrano la spinta, la direzione e il lancio di missili dell'incrociatore da battaglia di Klingon.

Quando un indicatore viene posto uguale a 1 in un sottoprogramma, si innesca un cambiamento in una delle astronavi. Per esempio, quando *fdav* è 1, il segmento di programma per l'aggiornamento della posizione aggiunge una piccola accelerazione (fra 2 e 5, a piacimento) all'accelerazione che in quel momento ha l'*Enterprise*. TREK deve poi riportare l'indicatore a 0.

Aggiornare la posizione di due astronavi e di un gruppetto di missili è molto più facile che gestire un ugual numero di stelle dotate di grande massa. La massa combinata delle macchine belliche è un'inezia a paragone della massa del sole centrale, quindi si assume che la mutua attrazione gravitazionale delle astronavi e dei missili sia pari a zero. Data la distanza di ciascun oggetto dal sole, TREK si limita a calcolare l'accelerazione dell'oggetto provocata dalla gravità che attrae in direzione centrale verso il sole, aggiorna la velocità dell'oggetto e infine modifica la sua posizione.

Anche un calcolo così semplice dal punto di vista concettuale richiede un notevole dispendio computazionale. Per ogni calcolo sono necessarie somme, prodotti e radici quadrate. Per evitare che l'eccesso di carico aritmetico rallenti troppo il gioco, TREK consulta una tabella; per ogni possibile distanza dal sole una matrice detta *forza* fornisce l'accelerazione predeterminata a cui è sottoposto un oggetto (*si veda l'illustrazione di questa pagina*). Dato che l'universo di Star Trek è toroidale, la nuova posizione ricavata dall'accelerazione fornita dalla tabella *forza* deve essere calcolata modulo la distanza orizzontale o verticale sul video rettangolare.

Due matrici, dette *vel* e *pos*, danno in ogni istante la velocità e la posizione delle due astronavi e di un massimo di 20 missili. Ciascuna matrice ha due colonne e 22 righe. Le prime due righe contengono i dati relativi alle astronavi e le altre 20 sono dedicate ai missili. Così nella prima riga di *vel* i due elementi *vel*(1,1) e *vel*(1,2) sono rispettivamente la velocità nella direzione x (orizzontale) e nella direzione y (verticale) dell'*Enterprise*. Analogamente, *vel*(2,1) e *vel*(2,2) danno le componenti di velocità reciprocamente perpendicolari dell'incrociatore di Klingon. Una speciale variabile detta *misnum* consente a TREK di tenere il conto dei missili in volo in un dato momento. *Misnum* va da 2 (nessun missile) a 22 (20 missili).

Mettendo insieme astronavi e missili nelle matrici si ottiene un programma più breve e leggermente più efficiente in quanto è necessario solo un ciclo per aggiornare le posizioni e le velocità. Prima dell'aggiornamento, però, sono necessarie due variabili per seguire l'orientamento dei due veicoli spaziali: *fdor* e *knor*. I loro valori sono espressi in gradi; l'angolo 0 indica un'astronave che punta verso est, 90 indica il nord, e così via. Ogni volta che un giocatore preme uno dei due tasti per la direzione, una delle due variabili viene aumentata o diminuita di 10 gradi secondo il caso.

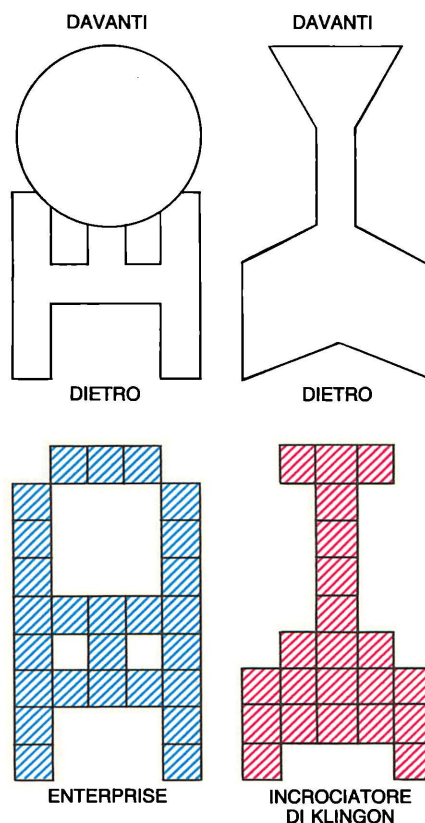
Il ciclo per l'aggiornamento della posizione ha come indice la lettera i, che va da 1 a 22. Per ogni valore di i, il programma calcola l'accelerazione: i valori in corso delle coordinate x e y dell'i-esima riga di *pos* sono elevati al quadrato e sommati. TREK trova poi la radice quadrata della somma e il numero viene arrotondato al più grande intero minore o uguale al numero stesso. La radice quadrata arrotondata è la distanza approssimata che intercorre tra il sole e la posizione dell'i-esimo oggetto. Quell'intero è poi preso come indice per la tabella di accelerazione, dove può essere cercata l'attrazione solare.

Si devono poi sommare per le due astronavi le componenti dell'accelerazione solare alle componenti della spinta. Se entrambi gli indicatori *fdav* e *knav* sono stati posti uguali a 1 (in altri termini, se è stato premuto il tasto della spinta) e nessuna delle due astronavi è a corto di combustibile, TREK deve moltiplicare la costante di spinta per le sue componenti orizzontale e verticale. Per il mio video, attribuisco alla costante di spinta il valore 3 e così ho una ragionevole manovrabilità per l'astronave. Per l'*Enterprise*, la componente orizzontale è pari a tre volte il coseno di *fdor*; quella verticale è tre volte il seno di *fdor*. Per l'astronave di Klingon le componenti di spinta si calcolano sulla base dell'angolo *knor*. Appena effettuato il calcolo, TREK riporta a zero *fdav* e *knav*; il propulsore viene spento fino alla successiva pressione del tasto della spinta.

Quando l'indice i è maggiore di 2, si può evitare il calcolo della spinta perché i missili non ne hanno. La parte restante del ciclo di aggiornamento è dedicata al calcolo delle nuove velocità e posizioni per gli oggetti che si muovono sullo schermo. Per ogni oggetto, si sommano il valore numerico dell'accelerazione a quello della velocità e il valore della velocità a quello della posizione. Si può avere un calcolo così semplice facendo in modo che la spinta e l'attrazione solare riflettano un sistema di misura che presuppone il passaggio di un'unità di

DISTANZA	FORZA
10	8,000
11	6,612
12	5,556
.	
.	
.	
178	0,025
179	0,025
180	0,025

$$\text{FORZA} = \frac{800}{(\text{DISTANZA})^2}$$

Esempio di tabella della forza e la sua formula

Icone* (in alto) *e loro versioni in pixel

tempo per ogni ciclo del programma.

Per controllare gli eventuali contatti tra i vari oggetti, il programma deve prima di tutto stabilire per ogni astronave se si trova su o all'interno del margine del sole. Dato che TREK ha già calcolato la distanza aggiornata tra ciascuna astronave e il sole, il programma deve solo confrontare quella distanza con il raggio solare, diciamo 10 unità. Se una delle due astronavi è entrata in collisione con il sole, TREK risponde con un adeguato messaggio sullo schermo, per esempio KLINGON VAPORIZZATO, poi passa al segmento di visualizzazione.

Un secondo controllo dei contatti deve stabilire, per ciascun missile, se si trova a una determinata breve distanza da una delle astronavi. Qui TREK utilizza una scorciatoia semplice ma efficace: trova la distanza tra le coordinate *x* del missile e di un'astronave, poi fa lo stesso con le coordinate *y*. Infine somma le due differenze; il procedimento evita le elevazioni al quadrato e l'estrazione di radici quadrate, e il risultato è preciso quasi quanto quello ottenuto con il consueto calcolo della distanza. Se la somma delle due differenze è minore di, diciamo, 4, il programma segna un contatto e sullo schermo appare un messaggio quale, per esempio, ENTERPRISE COLPITA DA UN MISSILE. VINCE KLINGON. Un unico ciclo effettua il controllo per ciascun missile. Il suo indice parte da 3 e termina a *misnum*, il numero di missili lanciati più 2.

Per aggiornare i livelli di energia delle astronavi il programma divide per 60 l'accelerazione solare ottenuta dalla tabella. Dato l'incremento di accelerazione, un'astronave che si avvicini al sole può ricevere un flusso più concentrato di fotoni solari. L'energia viene poi sommata a una variabile di combustibile detta, a seconda dell'astronave in gioco, *fdca* o *knca*. Ciascuna di queste variabili è diminuita di 0,1 quando si applica una spinta; le diminuzioni sono effettuate nel segmento di TREK dedicato all'aggiornamento della posizione. Quando un'astronave non ha più di un'unità di energia solare nel serbatoio, è considerata a corto di combustibile. Il serbatoio, all'inizio del gioco, contiene 10 unità di combustibile.

La gestione dei missili richiede una matrice detta *tempo* in cui è contenuto il numero di cicli di programma nella vita di ciascun missile. Quando un contatore di ciclo raggiunge 25, il missile corrispondente è tolto dalla matrice *pos*, il contatore è riportato a 0 e *misnum* è diminuito di 1. Il missile può essere tolto da *pos* in due modi. Il primo metodo è più facile da programmare ma può rallentare il gioco. TREK scorre la matrice a partire dal numero d'indice del missile rimosso e diminuisce di 1 il numero di riga di ciascun elemento. Così l'ultimo elemento da spostare si trova all'indice *misnum* e viene portato al numero di riga *misnum* −1. La stesssa operazione viene effettuata sulle matrici *vel* e *tempo*.

Una tecnica più veloce si avvale dell'osservazione che i missili più «vecchi» hanno gli indici più piccoli, in quanto sono stati i primi missili aggiunti all'elenco. Si può quindi seguire i missili senza spostare i loro indici; i missili che raggiungono un'età di 25 cicli di programma si devono trovare tutti all'inizio di un gruppo contiguo di missili in ciascuna matrice e sono i soli che devono essere tolti. Analogamente, i nuovi missili vengono sempre aggiunti alla fine del gruppo contiguo.

Si introducono altre due variabili, *vecchio* e *nuovo*, che servono da puntatori per i missili più vecchi e per quelli più recenti di ciascuna matrice. Quando si tolgono e si aggiungono missili, si devono cambiare solo i valori di *vecchio* e *nuovo*. Si può poi applicare l'aritmetica modulare per fare in modo che il gruppo contiguo di missili si sposti ciclicamente in ciascuna matrice. Quando si vuole aggiungere un nuovo missile all'indice 23, TREK riduce l'indice modulo 23 a 0 e poi aggiunge 3 per evitare di sostituire una coordinata di missile a una coordinata di astronave. Le variabili *vecchio* e *nuovo* subiscono lo stesso processo. Una struttura di dati di questo genere è detta «coda circolare». Se si utilizza questo trucco, si deve modificare il segmento del programma per l'aggiornamento della posizione: si deve dividere ogni singolo ciclo in due cicli più piccoli, uno per le astronavi e l'altro per i missili.

Quando un giocatore preme un tasto per lanciare un missile, TREK controlla innanzitutto il contatore di missili attivo in quel momento per quel giocatore. Se il contatore è minore di 10, TREK consulta i valori degli indicatori *fdfu* e *knfu*. Se, per esempio, *fdfu* è 1, il programma aggiunge 1 al contatore di missili per la Federazione, aumenta di 1 *misnum* e poi inserisce le coordinate di posizione e di velocità dell'*Enterprise* al giusto posto di *pos* e *vel*. Nel corso del procedimento, il programma dovrebbe aggiungere quattro unità alle coordinate di posizione e due unità alle coordinate di velocità; in entrambi i casi la somma è effettuata lungo la stessa direzione in cui si sta muovendo l'astronave in quel momento. Per esempio, la coordinata orizzontale di posizione di un missile lanciato dall'*Enterprise* è quattro volte il coseno di *fdor* più la coordinata orizzontale di posizione dell'*Enterprise*; invece la coordinata verticale aumenta di quattro volte il seno di *fdor*. La posizione iniziale del missile, quindi, è calcolata in modo da tenere il missile stesso alla larga dall'astronave, per impedire l'immediata distruzione di quest'ultima da parte del suo stesso missile fotonico. La stessa operazione applicata alle coordinate di velocità del missile riflette una velocità di lancio relativa pari a due unità per ciclo: un missile percorre, in un ciclo, due unità più dell'astronave da cui è stato lanciato.

L'ultima sezione principale di TREK visualizza il sole, le due astronavi e i missili attivi in un dato momento. Il programma disegna un cerchio di raggio 10 al centro dello schermo e poi percorre la matrice *pos*. Le astronavi della Federazione e di Klingon sono rappresentate da icone. La prima icona è sostanzialmente un cerchio che richiama la nota forma discoidale dell'*Enterprise*, con la sua coppia di propulsori. L'icona dell'astronave klingoniana è angolare. I lettori sono liberi di sbizzarrirsi con qualsiasi versione miniaturizzata di queste astronavi; basta rispettare due condizioni: che le due astronavi siano distinguibili una dall'altra e che si capisca in che direzione puntano. Nel disegnare l'icona per le due astronavi, TREK fa ricorso a una lista di punti di visualizzazione che devono essere traslati e ruotati in modo da rispecchiare la posizione e l'orientamento di ciascun oggetto. Per queste operazioni il programma consulta la matrice *pos* e le variabili *fdor* e *knor*.

I missili sono più semplici. In un singolo ciclo, consultando *pos*, il programma di visualizzazione disegna ciascun missile come un punto.

TREK è soggetto a uno svantaggio, dovuto alle caratteristiche degli schermi comunemente in uso. Questi schermi operano in modalità immagazzinamento: un oggetto disegnato sullo schermo vi rimane. Per evitare una gran confusione di astronavi e missili in vecchie posizioni, a ogni cambiamento di posizione, TREK deve disegnare ciascun oggetto due volte. Prima disegna in nero l'oggetto nella sua vecchia posizione (in pratica, quindi, cancellandolo) poi lo ridisegna nella nuova posizione nel colore normale.

Lascio i dettagli dell'inizializzazione del programma agli utenti che si cimenteranno con TREK. A dispetto della programmazione da videogioco, qualcuno di voi troverà forse troppo lenta l'azione. Per avere risultati migliori provate a compilare il vostro programma, oppure imponete una limitazione degli armamenti riducendo il numero di missili assegnato a ciascun giocatore.

Ho descritto solo la versione più spartana di Star Trek. Sono state costruite, e continuano a diffondersi, edizioni più fantasiose ma private: con tre o più astronavi, cannoni laser, grafica a colori e visualizzazione degli stati del gioco. Devo ringraziare Jonathan N. Groff di Clearwater, Florida, per avermi richiamato alla mente questo classico «clandestino» e per avermi illustrato una variante del gioco che prevede un'astronave klingoniana automatizzata. I terrestri che rappresentano la Federazione sono continuamente spazzati via dal programma di Groff.

In conclusione, desidero invitare tutti i lettori che saranno in grado di realizzare una versione funzionante di TREK a darmene notizia, fornendomi anche un preciso resoconto delle strategie da loro adottate per ottenere l'accelerazione del videogioco.

II. La sottile arte della simulazione

La simulazione è presente un po' in tutti i giochi proposti nelle «(Ri)creazioni al calcolatore», ma in questa serie di articoli risulta tema dominante, a vari livelli di astrazione. Si va da una simulazione di tipo «ecologico» come quella del pianeta Wa-tor con le sue popolazioni e i suoi delicati equilibri, a un immancabile simulatore di volo, per finire con un gioco astratto come «Vita» e con i suoi rapporti con il più astratto fra i modelli delle macchine che elaborano informazioni, la macchina di Turing.

Vita, inventato dal matematico inglese John Horton Conway, è ormai un classico, da quando nel 1971 Martin Gardner lo rese di pubblico dominio nella sua rubrica di «Giochi matematici» pubblicata da «Le Scienze». Si tratta di un gioco senza giocatori (un gioco «a zero») che si svolge su una matrice infinita di celle (o cellule) quadrangolari. A ogni istante alcune celle sono vive, le altre sono morte: a chi imposta il gioco è lasciato il compito di definire le celle vive all'istante iniziale, dopodiché l'evoluzione è determinata esclusivamente dalle regole di Vita, che specificano quali celle passino da uno stato all'altro, in funzione dello stato delle celle che le circondano.

Le regole sono semplicissime e il pregio del gioco sta tutto nella complessità e nella varietà di configurazioni che si possono generare a partire da diverse disposizioni iniziali. Stabilire se esista un criterio generale per prevedere il destino di una configurazione di Vita (se svanirà completamente, se diventerà statica od oscillante, se viaggerà nel piano, o si espanderà indefinitamente, per esempio) è una sfida notevole.

Come si può vedere negli articoli di questa parte, non è difficile programmare un calcolatore perché giochi a Vita e permetta di esplorarne senza troppa fatica le infinite combinazioni; ma la sfida più attraente è quella di scoprire configurazioni che possano simulare un calcolatore digitale. Gli elementi ci sono tutti, e se ne possono trarre, come si vedrà, conseguenze interessanti. Per i più appassionati, segnaliamo l'esistenza di un bel capitolo dedicato a Vita nel secondo volume di *Winning Ways*, scritto da Conway con Berlekamp e Guy (Academic Press, 1982), dove gli autori mettono alla prova l'idea di programmare il calcolatore simulato da Vita per cercare la soluzione a un problema arbitrariamente difficile...

Squali e altri pesci combattono una guerra ecologica sul pianeta Wa-Tor

di A. K. Dewdney

Le Scienze, febbraio 1985

Da qualche parte, in una direzione che si può solo chiamare ricreativa e a una distanza limitata soltanto dal proprio valore programmatorio, il pianeta Wa-Tor nuota tra le stelle. È un pianeta a forma di toro, cioè di ciambella, ed è interamente coperto dall'acqua. Gli abitanti principali di Wa-Tor sono squali e altri pesci, così chiamati perché queste sono le creature della Terra a cui maggiormente assomigliano. Gli squali di Wa-Tor mangiano i pesci più piccoli, di cui sembra vi sia sempre grande abbondanza.

Questo semplice ecosistema potrebbe apparire stabile, quasi soporifico, se non fosse per il fatto che le popolazioni degli squali e degli altri pesci subiscono violente oscillazioni. Molte volte, nel passato, la popolazione di questi ultimi è stata quasi interamente divorata, altre volte gli squali hanno sfiorato l'estinzione (anche quando gli altri pesci abbondavano). Eppure, sia gli squali sia gli altri pesci sopravvivono. Per capire perché, ho scritto un programma che simula le loro attività alimentari e riproduttive.

Prima di mostrare questi ritmi ecologici sullo schermo (*si veda l'illustrazione della pagina successiva*), ho meditato a lungo sulle regole e i particolari del programma WATOR. Un giorno, a pranzo, mi trovai a chiacchierare con David Wiseman, il mago dei sistemi di calcolo del mio dipartimento all'Università dell'Ontario occidentale. Dopo avergli descritto il progetto, notai che Magi (così viene chiamato Wiseman) sorrideva in modo enigmatico. Il mattino seguente, tutto fiero, mi fece entrare nel suo ufficio per mostrarmi un programma già pronto a funzionare.

«Guarda», disse, e premette un tasto. Un assortimento inizialmente casuale di squali e altri pesci guizzava da un punto all'altro in modo apparentemente caotico. Alcuni squali non riuscivano a nutrirsi e scomparivano; altri avevano una prole vorace quanto loro. Alcuni pesci più piccoli, abbastanza fortunati da occupare una regione in cui in quel momento non c'erano squali, si moltiplicavano formando un grande banco. A questo punto, un gran numero di squali scoprivano il banco, si raggruppavano ai suoi lati e vi penetravano per un poco facendosi strada mangiando. Pochi minuti dopo, la situazione sullo schermo di Magi era la seguente: 578 pesci piccoli e solamente 68 squali.

Qualcuno entrò nell'ufficio di Magi e corse subito fuori. Nemmeno cinque minuti dopo, la stanza era piena di gente che faceva il tifo per gli squali. A poco a poco, i piccoli pesci furono circondati da un muro di squali, mentre in un altro punto dello schermo un banco poco numeroso di piccoli pesci si moltiplicava senza essere notato. Mormorii si levarono quando infine questi pesci scomparvero e gli squali, agitandosi alla ricerca di preda, incominciarono a morire uno a uno. Mi venne l'idea di modificare le regole per consentire agli squali di mangiarsi l'un l'altro, ma mi resi conto che un'alimentazione parossistica non avrebbe prolungato in modo significativo la loro esistenza e avrebbe potuto mettere a repentaglio la vita dell'altro piccolo banco. Quando infine due squali, nel loro vagare, si imbatterono in esso, il ciclo ricominciò.

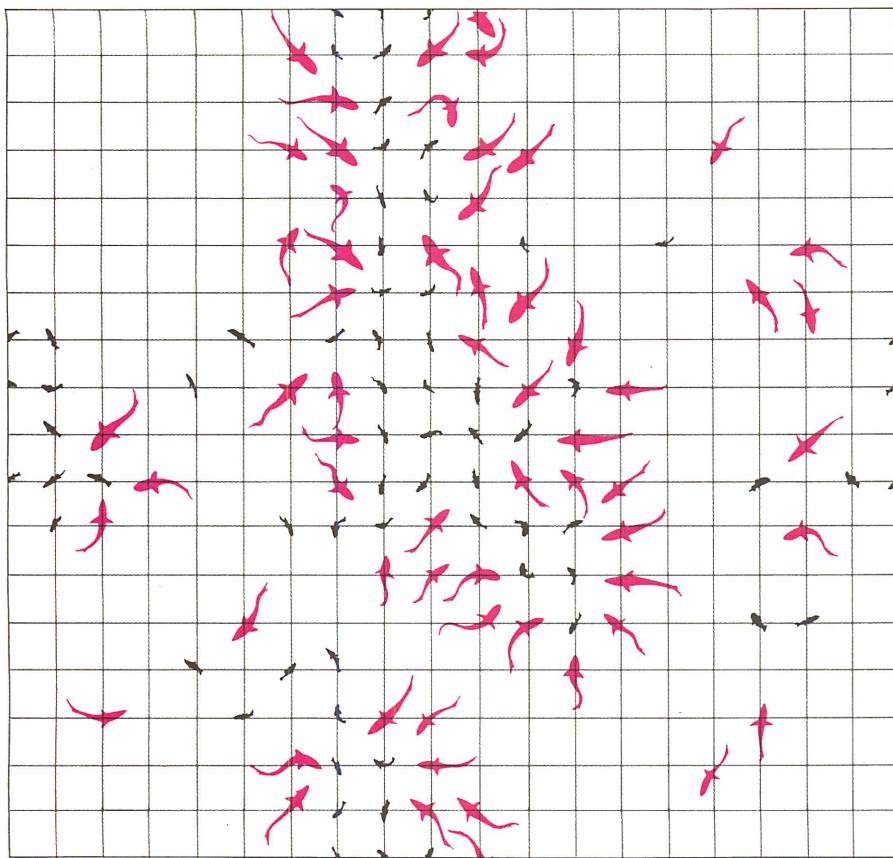
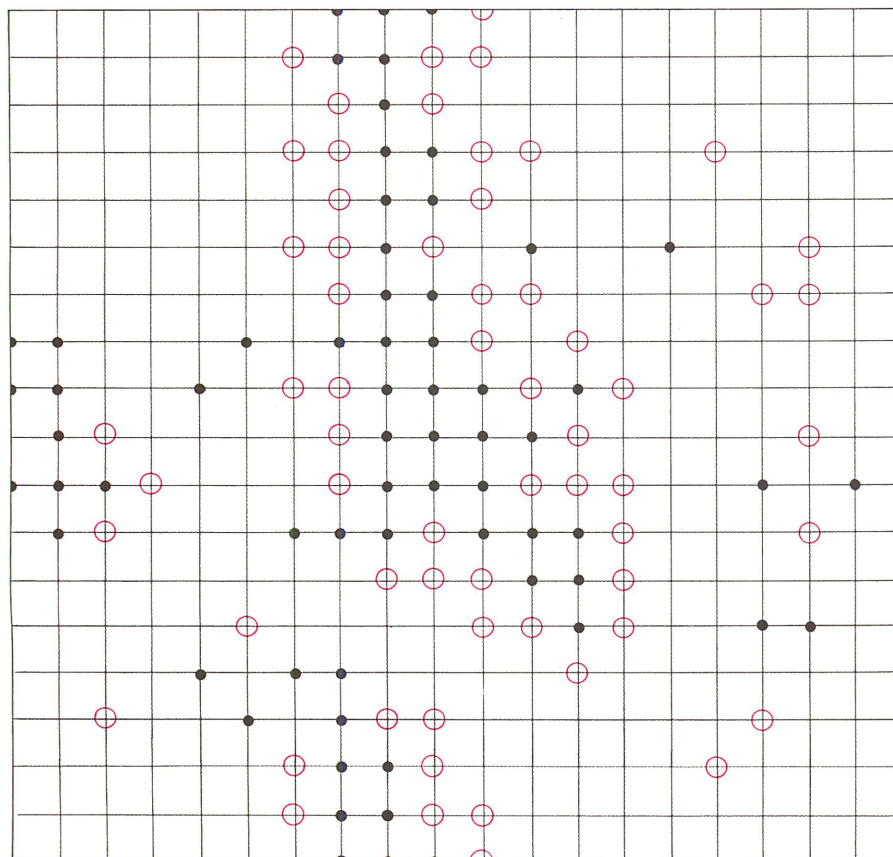
Il programma di Wa-Tor non è né molto lungo né difficile da scrivere. I lettori che possiedano un calcolatore personale, anche con poca esperienza di programmazione, si sentiranno gratificati quando infine il codice sarà stato scritto, corretto e fatto girare. Parametri quali i tempi di alimentazione, i periodi di carestia e la dimensione delle popolazioni iniziali possono essere stabiliti in principio; poi, non c'è che da mettersi comodi e osservare come un miscuglio inizialmente disorganizzato di squali e pesci più piccoli assuma a poco a poco configurazioni ecologiche.

Nel programma WATOR compaiono molte regole semplici che governano il comportamento sia degli squali sia degli altri pesci. Queste creature nuotano in un oceano a griglia rettangolare i cui lati opposti sono identificati a due a due. Questo significa semplicemente che se uno squalo o un altro pesce occupa un punto della griglia a destra e decide di nuotare verso est (cioè verso destra), riapparirà nel corrispondente punto della griglia a sinistra. Lo spazio bidimensionale senza confini che ne risulta è un toro, l'effettiva superficie di Wa-Tor (*si veda l'illustrazione di pagina 55*). Per scrivere il proprio programma WATOR si può scegliere una qualsiasi opportuna dimensione per la griglia oceano. Per esempio, Magi, il cui programma gira su un calcolatore VAX, ha definito un oceano largo 80 punti e alto 23. La mia versione di WATOR, scritta per un IBM PC, usa un più modesto oceano 32 per 14.

Il tempo trascorre con salti discreti, che chiamerò crononi. Durante ogni cronone uno squalo o un altro pesce può spostarsi in direzione nord, est, sud o ovest verso un punto adiacente, sempre che il punto non sia già occupato da un membro della sua stessa specie. La scelta è effettuata da un generatore di numeri casuali. Per un pesce non predatore la scelta è semplice: scegliere a caso un punto libero adiacente e spostarcisi. Se tutti i quattro punti adiacenti sono occupati, il pesce non si muove. Dato che predare i pesci ha la priorità sul semplice movimento, le regole per uno squalo sono più complesse: scegliere a caso uno dei punti adiacenti occupati dalle eventuali prede, spostarcisi e divorarle. In loro mancanza, lo squalo si sposta esattamente come un pesce non predatore, evitando gli altri squali.

Il creatore di WATOR sceglie 5 parametri per avviare una data simulazione. I parametri *npesci* e *nsquali* rappresentano il numero di pesci (possibili prede degli squali) e di squali all'inizio dell'esecuzione. Il programma distribuisce i pesci e gli squali a caso e in modo più o meno uniforme sulla superficie del pianeta. I parametri *pprole* e *sprole* designano il numero di crononi che devono passare prima che, rispettivamente, un pesce e uno squalo abbiano un figlio. (Entrambe le specie sono evidentemente partenogeniche.) Infine *inedia* indica il numero di crononi che uno squalo ha a disposizione per trovare di che cibarsi. Se lo squalo non riesce a mangiare prima che sia trascorso questo tempo, muore e scompare dalla circolazione. Durante ogni cronone, WATOR muove una volta ciascun pesce e ciascuno squalo e mostra i risultati sullo schermo.

Io e Magi abbiamo assistito a un gran numero di sceneggiature a cinque parametri in cui l'oceano di Wa-Tor si sovrappopolava di pesci finché gli squali si moltiplicavano al punto da mangiare tutti gli altri pesci e poi morire. In altre occasioni abbiamo visto divorati tutti i pesci di un grande banco; gli squali che si erano ingozzati finivano col morire di fame, senza scoprire mai un gruppetto di pesci lì vicino. In qualche caso abbiamo visto la relazione preda-predatore conservarsi per due o anche tre cicli prima del definitivo tracollo nella popolazione degli squali. Nulla, però, nei parametri scelti per quelle sceneggiature dava qualche suggerimento sulle caratteristiche che garantirebbero un ecosistema eterno. Come avevano fatto a sopravvivere gli abitanti di Wa-Tor?

Qualcuno ha detto che biologia vuol dire destino. Magi e io siamo tentati di dichiarare che ecologia vuol dire geometria, almeno per quanto riguarda il pianeta Wa-Tor. Il definitivo destino di una data sceneggiatura non sembra dipendere dalla distribuzione casuale iniziale di un certo numero di squali e di altri pesci. E neppure sembra dipendere in modo accidentale dall'effettivo movimento casuale degli squali e di quei pesci. La probabilità del tracollo di una popolazione sembra invece seguire strettamente la geometria

Un'immagine realistica di squali che predano pesci

Un'immagine più facilmente programmabile, in cui i cerchi rappresentano gli squali e i punti i pesci che essi predano

pesce-squalo che si manifesta sui nostri schermi: più cresce l'organizzazione e la localizzazione delle due popolazioni, più è probabile la definitiva condanna dell'ecosistema. Meditando su questo argomento, ci siamo chiesti come avremmo potuto scegliere i cinque parametri in modo da rompere la geometria. Ci venne allora un lampo di intuizione: nel caso gli squali si fossero radunati ai lati di un banco di pesci, un modo per spezzare la geometria risultante sarebbe potuto consistere nel diminuire la frequenza di riproduzione degli squali. L'assembramento stesso, dopo tutto, derivava più dalla riproduzione che dal movimento.

Prima di formulare questa ipotesi, avevamo scelto tempi di riproduzione grosso modo uguali per gli squali e gli altri pesci. L'equilibrio nei ritmi riproduttivi, pensavamo, avrebbe dato come conseguenza un equilibrio nelle popolazioni. Questo genere di ragionamento vago è probabilmente all'origine di molte calamità dell'attuale mondo tecnologico. In ogni caso, io misi 20 squali e 200 altri pesci nel mio oceano 32 per 14 e stabilii che i pesci si riproducessero ogni tre crononi, mentre gli squali si riproducevano solo dopo 10 crononi. Il tempo di morte per inedia degli squali venne fissato, in modo più o meno arbitrario, in tre crononi. Fummo ricompensati, dopo aver osservato per 15 minuti il mio programma, piuttosto lento, dal vedere che la popolazione si ricostituiva dopo il declino iniziale. Per di più, la geometria, pur essendo ancor presente, era più suggerita che definita. I banchi erano conglomerati informi con margini sfilacciati e sia gli squali sia gli altri pesci si aggiravano casualmente in alcuni punti dello schermo.

Lasciai girare il programma tutto il pomeriggio, dando qualche occhiata di tanto in tanto mentre mi occupavo di questioni più importanti. Il programma girò tutta la notte e quando entrai nel mio ufficio, dopo la lezione del mattino, trovai che sia gli squali sia gli altri pesci continuavano la loro esistenza ciclica. Questa volta era proprio Wa-Tor!

Ci sono molti modi per implementare un programma WATOR, ma forse il più semplice consiste nell'utilizzare matrici bidimensionali. Io uso cinque matrici a cui possiamo dare il nome di PESCI, SQUALI, MOSSAPESCE, MOSSASQUALO e INEDIA. Queste matrici, tutte di 32 per 14, registrano le posizioni e le età degli squali e degli altri pesci. PESCI (I, J) rappresenta la presenza o l'assenza di un pesce nel punto con coordinate (I, J). Se non c'è un pesce, la posizione ha il valore −1. In caso contrario, contiene una registrazione dell'età in crononi del pesce che vi si trova. Lo stesso schema viene usato nella matrice SQUALI per registrare le posizioni e le età degli squali. La matrice MOSSAPESCE registra, per ogni posizione, se un pesce è stato spostato in quella posizione durante il ciclo computazionale in corso. Questa registrazione impedisce al programma di muovere un pesce due volte durante lo stesso cronone. MOSSASQUALO adempie la stessa fun-

zione per gli squali. La matrice chiamata INEDIA registra il momento in cui uno squalo ha mangiato per l'ultima volta. Se in una posizione non ci sono squali, il valore corrispondente è −1.

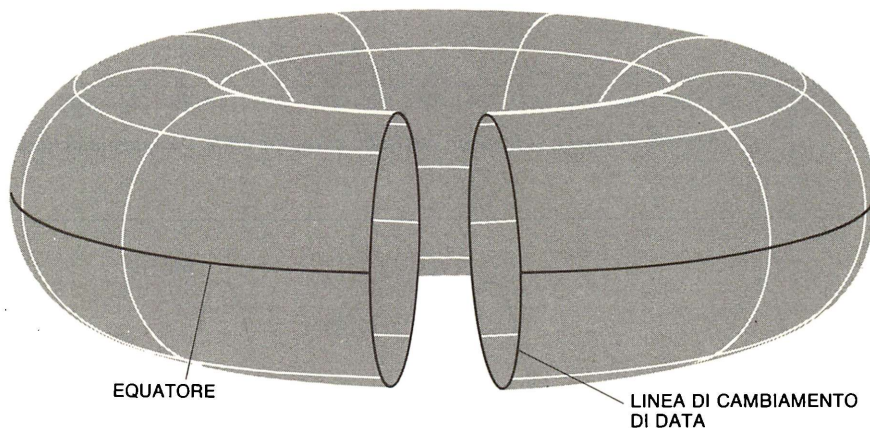
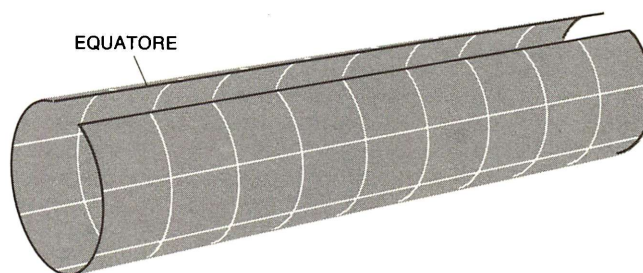
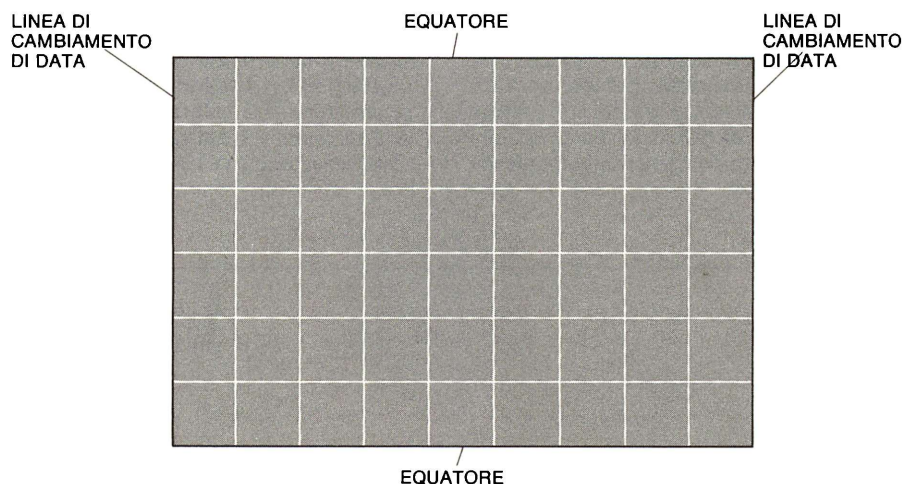
Il modo più semplice per visualizzare ciò che avviene su Wa-Tor è una linea di caratteri sullo schermo per ogni riga delle matrici; un vuoto in una posizione significa che non è occupato. Un punto (.) rappresenta un pesce e uno zero (0) rappresenta uno squalo. Pur essendo apparentemente limitata, questa visualizzazione è sorprendentemente ricca di informazione e piacevole da guardare.

Nella fase iniziale di WATOR, i pesci, eventuali prede degli squali, e gli squali nel numero richiesto sono distribuiti uniformemente nell'oceano toroidale. Il programma, poi, attraversa ciclicamente i tre segmenti o sottoprogrammi descritti qui di seguito; ogni ciclo del programma dura un cronone di tempo.

I pesci nuotano e si riproducono:
Per ogni pesce della matrice PESCI, il programma elenca le posizioni adiacenti non occupate e sposta il pesce su una di esse a caso. PESCI allora deve essere posta a −1 nella vecchia posizione e all'attuale età del pesce nella nuova posizione. La matrice MOSSAPESCE viene aggiornata nel modo descritto prima. Se l'età del pesce è uguale a *pprole*, il programma mette un nuovo pesce nella vecchia posizione e dà età 0 a entrambi i pesci. MOSSAPESCE registra il nuovo pesce. Se tutte le posizioni adiacenti sono occupate, il pesce non si muove né si riproduce.

Gli squali predano e si riproducono:
Per ogni squalo della matrice SQUALI, il programma elenca le posizioni dei pesci adiacenti (se ce ne sono). Lo squalo sceglie a caso una di queste posizioni, vi si sposta e mangia il pesce. Questo non significa solo che il programma deve modificare SQUALI e MOSSASQUALO come ha modificato PESCI e MOSSAPESCE, ma anche che deve assegnare −1 alla corrispondente posizione nella matrice PESCI. Inoltre, a INEDIA in quella posizione è assegnato il valore 0. Se non ci sono pesci da predare, lo squalo si muove come uno di questi pesci. Se l'età dello squalo è uguale a *sprole*, nasce un nuovo squalo esattamente come nasce un nuovo pesce.

Visualizzazione:
Il programma esamina la matrice PESCI e la matrice SQUALI, poi visualizza un punto per ogni pesce e un cerchio per ogni squalo. La visualizzazione può essere effettuata tutta insieme oppure può essere divisa in due parti: una eseguita dopo che si sono spostati i pesci, l'altra dopo che si sono spostati gli squali.

Per popolare l'oceano iniziale, il programmatore costruisce un ciclo che genera due numeri casuali per *npesci* volte. I numeri sono in proporzione alle dimensioni orizzontale e verticale dell'oceano che si vuole avere. In ognuna delle posizioni casuali così scelte, il programma mette un pesce nella matrice PESCI e gli assegna un'età compresa tra 0 e *pprole*. La distribuzione degli squali avviene in modo analogo. In entrambi i casi, si verifica se la posizione è già occupata. Dando età casuali sia agli squali sia agli altri pesci, si ottiene che essi si riproducano in momenti casuali, con un effetto di maggior naturalezza. Senza questa precauzione, si assi-

Il pianeta toroidale Wa-Tor e la sua rappresentazione su una carta piana (o sullo schermo piano di un calcolatore)

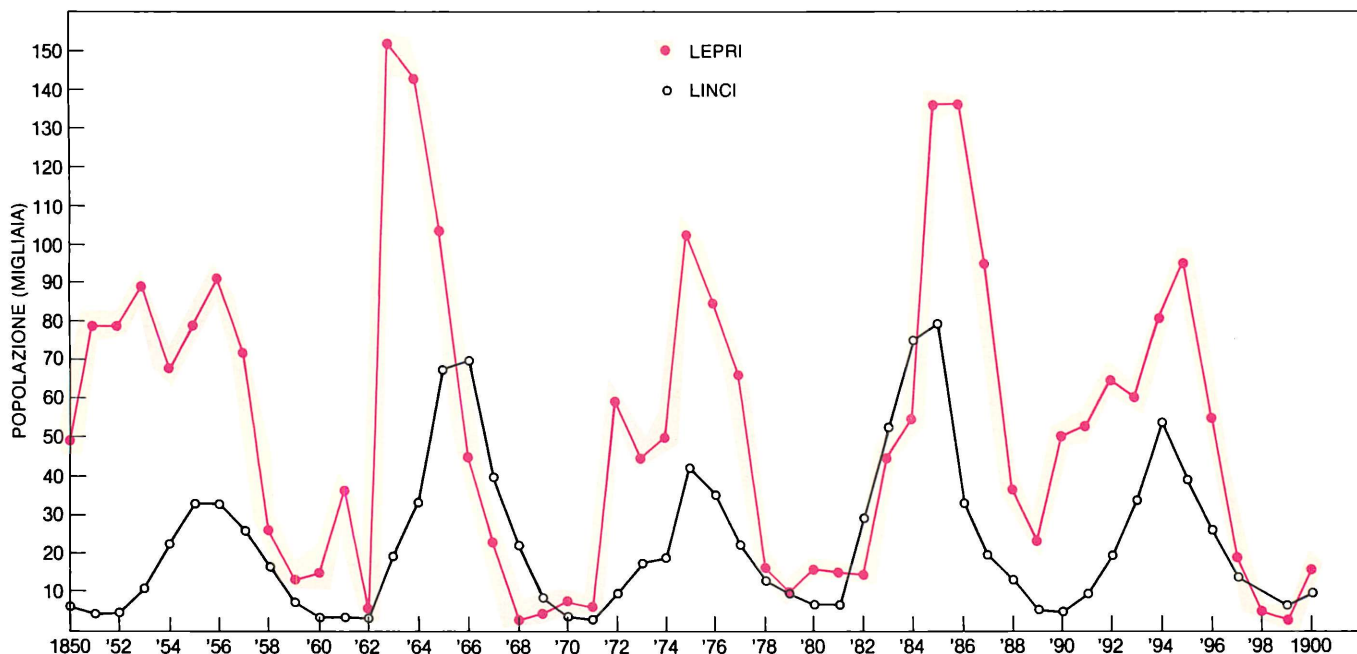
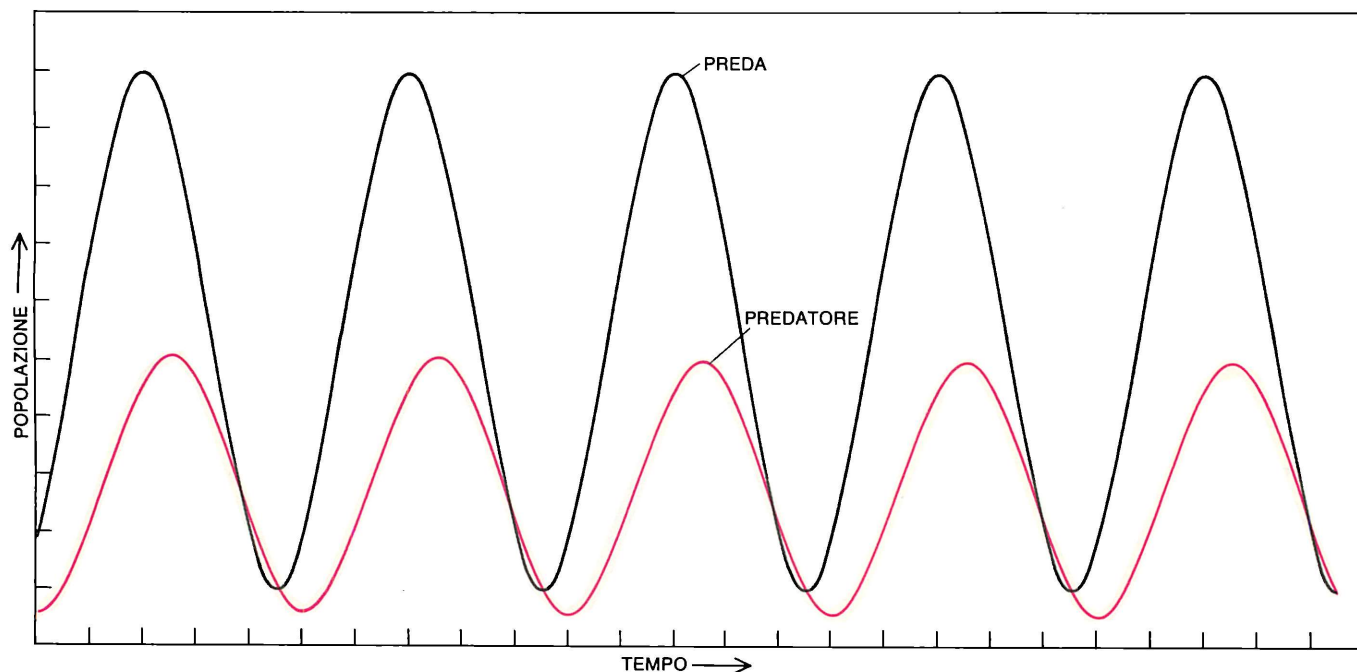
sterebbe a un improvviso raddoppio del numero degli squali e dei pesci, cosa sconcertante e innaturale.

Forse i programmatori alle prime armi possono trovare la precedente descrizione un po' troppo generale per farsi un'idea chiara di come scrivere un programma WATOR. Possono iniziare allora a scrivere quello che si può chiamare programma dell'ubriaco barcollante. Un programma di questo genere potrebbe essere formato da un unico ciclo (per esempio un ciclo *while*) con sette istruzioni. Le istruzioni qui sono scritte in un generico linguaggio algoritmico. Gli assegnamenti sono indicati da una freccia rivolta verso sinistra e le variabili X e Y sono le coordinate di un ubriaco barcollante. Esse variano a seconda dell'intero casuale assegnato a una variabile *direzione*. A seconda che questo intero sia uguale a 0, 1, 2 o 3, l'ubriaco (un punto sullo schermo) si muove verso nord, est, sud o ovest.

direzione ← parte intera
  di (*random* × 4)
se *direzione* = 0 allora $X \leftarrow X + 1$
se *direzione* = 1 allora $X \leftarrow X - 1$
se *direzione* = 2 allora $Y \leftarrow Y + 1$
se *direzione* = 3 allora $Y \leftarrow Y - 1$
visualizzazione (X, Y)

Se il vostro generatore di numeri casuali produce un numero decimale *casuale* compreso tra 0 e 1, questo algoritmo lo ridurrà a un numero decimale di valore compreso tra 0 e 0,3999. La parte intera del numero deve essere 0, 1, 2 o 3.

Non posso certo affermare che stare a guardare un punto luminoso che vaga sullo schermo sia paragonabile al dramma ecologico degli squali e dei pesci che essi predano, ma scrivendo questo programma si può avere un'idea di come si potrebbero costruire le varie parti di WATOR.

I programmatori esperti che leggano questo articolo penseranno ad altri modi

Numero di linci e di lepri (in migliaia) catturate dalla Compagnia della Baia di Hudson dal 1850 al 1900

Una relazione teorica predatore-preda: una soluzione alle equazioni di Lotke-Volterra

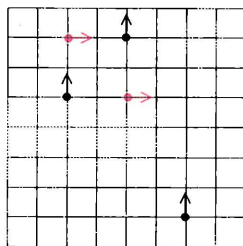
per scrivere il programma WATOR. La quantità di elaborazione può essere ridotta di molto usando liste concatenate per seguire gli squali e i pesci che essi predano.

WATOR può fornire qualche indicazione sulle popolazioni animali che vivono sulla Terra. Sappiamo che le piccole popolazioni hanno un'alta probabilità di estinzione e, anche se né i predatori né la preda scompaiono, quasi certamente essi sottostanno a cicliche variazioni nel numero. In semplici ecosistemi predatore-preda, le popolazioni dei predatori e delle prede seguono due cicli sovrapposti di massimi e di minimi di popolazione. Le dimensioni delle popolazioni di lepri e di linci registrate dalla Compagnia della Baia di Hudson dal 1847 al 1903 nel territorio subartico canadese seguono questo schema (*si veda l'illustrazione della pagina a fronte*). Le cifre indicano il numero di animali delle due specie catturati di anno in anno. È presumibile che questi numeri siano proporzionali alle effettive dimensioni delle due popolazioni durante questo periodo. Se lo sono, i cicli si spiegano facilmente come il risultato delle predazioni operate dalle linci in una popolazione sempre crescente di lepri, popolazione che comincia a declinare con l'aumento del numero di linci. Ben presto comincia a esservi meno cibo per le linci, che iniziano a morire di fame o a riprodursi di meno (o tutt'e due le cose). Quando il numero delle linci si riduce, le lepri iniziano di nuovo a moltiplicarsi.

Messo a confronto con questo diagramma è un insieme di curve che rappresentano una soluzione alle equazioni di Lotke-Volterra. Queste equazioni sono state formulate per la prima volta nel 1931 dal matematico Vito Volterra. Esse presuppongono quello che si potrebbe chiamare un predatore continuo, cioè continuamente a caccia di una preda continua. Le soluzioni di queste equazioni mostrano una variazione ciclica che, a prima vista, sembra riprodurre i dati empirici relativi alle lepri e alle linci. I biologi, però, non concordano sul fatto che l'entità delle lepri e delle linci sia spiegabile con un ragionamento così semplice. In primo luogo, sono in gioco almeno altri due nemici delle lepri: i microbi e l'uomo.

È del tutto sensato, comunque, compilare statistiche degli squali e degli altri pesci di Wa-Tor, e io e Magi l'abbiamo fatto. I nostri grafici più recenti relativi alle popolazioni degli squali e dei pesci tendono ad assomigliare ai diagrammi delle linci e delle lepri più di quanto facciano le soluzioni di Lotke-Volterra. Eppure, continua a lasciarci perplessi l'instabilità mostrata sul lungo periodo da talune combinazioni di parametri. Forse qualche lettore, lavorando sul suo programma WATOR, potrà dare qualche ulteriore illuminazione. Esiste qualche regola generale che permetta di prevedere, data una combinazione di parametri, se l'ecosistema risultante sarà stabile? In che misura le fluttuazioni cicliche seguono le equazioni di Lotke-Volterra?

L'oceano di Wa-Tor è toroidale per una ragione semplicissima: è molto più semplice scrivere un programma per un oceano che non abbia limiti o coste. Se l'oceano deve essere largo, per esempio, 32 unità, è facile usare numeri modulo 32 come coordinate X dei pesci e degli squali. Se questi hanno come coordinata X 31 e appaiono sul margine destro dello schermo durante un cronone, possono anche avere come coordinata $X\ 32 = 0$ e apparire sul margine sinistro nel cronone successivo. Lo stesso per l'altra coordinata.

L'oceano toroidale di Wa-Tor dà luogo ad alcuni effetti molto strani. Il primo ha a che fare con un difetto di una delle prime versioni del mio programma WATOR. Questo difetto faceva sì che ogni pesce nuotasse di un'unità verso nord e ogni squalo nuotasse di un'unità verso est durante ogni cronone di tempo. Uno squalo, quindi, arrivava a catturare un pesce solo se si trovava a occupare la stessa posizione della sua preda. Dato l'oceano qui sotto, quanti pesci non venivano mai catturati da uno squalo?

Un altro rompicapo prevede squali e pesci intelligenti. Supponiamo che a ogni mossa ogni squalo e ogni pesce si spostino su uno qualsiasi dei loro quattro punti adiacenti. Ne risulta che un singolo pesce, se è abbastanza intelligente, può sempre sfuggire a un singolo squalo, per quanto intelligente sia il predatore. Nell'oceano toroidale di Wa-Tor, due squali a caccia di un pesce isolato possono produrre due risultati differenti. Se dotate ogni creatura di tutta l'intelligenza che volete, arrivando anche a permettere che gli squali cooperino nella caccia, riuscite a scoprire un modo in cui il pesce possa cavarsela? Il risultato non dipende dalle dimensioni dell'oceano.

Soluzioni proposte

In linea generale, la scelta dei giusti parametri ha prodotto notevoli fluttuazioni nelle popolazioni degli squali e dei pesci. Alcuni, desiderosi di rendere Wa-Tor più simile alla Terra, hanno aggiunto particolari caratteristici ai loro programmi. Il gioco, in effetti, invita a una sua complicazione, che è certamente benvenuta. L'introduzione di un sistema variante, però, presenta lo svantaggio (a parità di altri fattori) di rendere ardui i confronti con il sistema standard.

I costruttori di un sistema iniziale si sono trovati ad affrontare svariate questioni, tra le quali la durata della sopravvivenza. Chiaramente, non esiste alcun problema per le popolazioni eterne, ma sarebbe utile avere una misura delle sceneggiature che non sono eterne. Misurare con i crononi può essere fuorviante quando si scelgono la durata delle estensioni di vita e i tempi di riproduzione per gli squali.

Anche la misurazione per cicli solleva problemi: che cos'è un ciclo? È divertente osservare che se squali e pesci sopravvivono a un sufficiente numero di ripetizioni del ciclo base con numeri casuali, si ripeterà una configurazione precedente, in accordo con il ciclo, e da lì in avanti alle popolazioni sarà ovviamente garantita la vita eterna.

Numerosi sono anche coloro che hanno descritto teorie moderne utili all'analisi di Wa-Tor. Non è stata ancora detta l'ultima parola a proposito del dilemma se le matrici stocastiche ci metteranno in grado di derivare specifiche probabilità di sopravvivenza da combinazioni arbitrarie di parametri o no. È interessante notare, però, che le equazioni di Lotka-Volterra (dalla loro formulazione nel 1931) sono state elaborate in modo da prendere in considerazione la diffusione come fattore che riguarda sia il predatore sia la preda. La diffusione fa assumere forme più complesse alle soluzioni di Lotka-Volterra, che di solito variano con regolarità.

Milton Boyd di Amherst, nel New Hampshire, ha sfruttato un diagramma di fase per analizzare la dinamica delle popolazioni di squali e altri pesci. A ogni istante t, si riportano in grafico in numero x dei pesci e quello y degli squali come coordinate di un singolo punto. Mentre il tempo avanza e le popolazioni seguono il loro ciclo, il punto descrive un'orbita erratica intorno a un occhio, o centro, fisso. Con questa tecnica, Boyd ha studiato l'effetto delle dimensioni dell'oceano sulla sopravvivenza.

Tra le innovazioni introdotte possiamo annoverare una forza vitale degli squali, mutazioni, doppie popolazioni di pesci e plancton.

Vorrei aggiungere che i pesci comuni di Wa-Tor si cibano di un plancton oceanico sparso ovunque in abbondanza. Questa caratteristica è stata resa esplicita facendo in modo di mettere il plancton in ogni punto non occupato da uno squalo o da un altro pesce. Il plancton prolifica in punti altrimenti vuoti e ha con i pesci comuni la stessa relazione che i pesci comuni hanno con gli squali. Anche in questo caso esistono popolazioni eterne.

Nessuno è riuscito a risolvere il problema dell'inseguimento toroidale. Rivelerò solo metà della soluzione, in modo da riservare ai lettori il piacere di trovare l'altra metà. Si ricordi che, a ogni turno, il pesce si muove e poi si muovono i due squali. Come in Wa-Tor, non è consentito rimanere nello stesso punto. Ogni raggio segue una diagonale e gira intorno al toro, ricongiungendosi presto o tardi con se stesso. Quando entrambi gli squali occupano una coppia di raggi opposti, non importa in che modo il pesce si muove: uno squalo insegue a distanza costante e l'altro squalo si avvicina. Il pesce è condannato. Lascio ai lettori scoprire in che modo gli squali, per così dire, vadano alla caccia dei raggi.

Un calcolatore usato come telescopio per incontri ravvicinati con ammassi stellari

di A. K. Dewdney

Le Scienze, marzo 1986

Nella profondità dello spazio, un ammasso stellare esegue una danza cosmica sulla melodia della gravità. Su un arco di tempo paragonabile alla durata della vita di un uomo, le stelle si muovono appena; ma in un periodo più lungo, in cui gli anni sono equivalenti a secondi, intrecciano con le loro orbite figure complesse. Di tanto in tanto, una stella incontra una sua vicina in un *pas de deux* che la lancia nello spazio. Se queste fughe sono più che occasionali, l'ammasso a poco a poco si contrae e il nucleo inizia a collassare.

Un telescopio di grande potenza può rivelare la struttura di qualche ammasso della nostra galassia, ma non può comprimere gli anni in secondi; solo un calcolatore arriva a tanto. Un calcolatore può anche essere programmato in modo da diventare una sorta di telescopio per osservare ammassi ipotetici. A velocità cosmica, si può vedere il movimento dei membri di un ammasso come se fosse una successione di istantanee in cui ogni stella lascia una traccia punteggiata che serpeggia attraverso l'ammasso *(si veda l'illustrazione della pagina a fronte)*.

Bastano le forze gravitazionali per spiegare l'evoluzione che gli astronomi deducono dagli ammassi osservati? I calcolatori ci aiutano a trovare risposte per questa e altre domande correlate. Alla Princeton University, nel maggio 1984, si è svolto un convegno di esperti in simulazioni e di teorici per discutere sulla consistenza degli ammassi stellari ipotetici e reali. Si trattava del 113° simposio della International Astronomical Union, interamente dedicato alla dinamica degli ammassi stellari.

È relativamente facile definire la coreografia di un balletto cosmico. In linea di principio, le interazioni stellari all'interno di un ammasso sono di una semplicità classica: entrambi i membri di una coppia di stelle sono sottoposti a una forza gravitazionale proporzionale all'inverso del quadrato della distanza che le separa. La forza, a sua volta, è proporzionale al prodotto delle due masse stellari. È una formula facile da calcolare: si moltiplicano le masse; poi si moltiplica il prodotto per una costante di proporzionalità e si divide per il quadrato della distanza tra le due stelle. La somma totale di tutte le coppie di forze che agiscono nel corso del tempo determina presumibilmente la configurazione di movimenti all'interno dell'ammasso. Un programma chiamato CLUSTER (ammasso) calcola la somma delle forze per ogni stella e sposta la somma dalla sua posizione attuale a una posizione vicina. Questa operazione viene compiuta ripetutamente per secoli di tempo simulato.

È abbastanza noioso battere alla tastiera le coordinate e le velocità di molte stelle, ma una volta svolto questo compito si può seguire dalla poltrona il dispiegarsi dell'universo sullo schermo. Le stelle al centro dell'ammasso seguono percorsi erratici e oscillanti; quelle alla periferia se ne vanno alla deriva, poi si fermano e scivolano indietro. Tra gli eventi più interessanti ci sono gli incontri ravvicinati e le fughe.

Quando due stelle si avvicinano molto, si impartiscono a vicenda una tremenda spinta gravitazionale e quindi si allontanano velocemente. Di solito le fughe sono il risultato di uno o più incontri ravvicinati. Quando una stella si allontana dal suo ammasso, ci sono solo due possibilità: o ritorna o non ritorna. La velocità di fuga di un corpo astronomico dipende dalla sua massa e dalla massa del corpo o dell'oggetto da cui fugge. Se la velocità viene raggiunta da una stella che si muove verso l'esterno del suo ammasso, la stella non tornerà mai. Gli appassionati che si dedicano per la prima volta alla simulazione di un ammasso hanno una buona probabilità di assistere a numerose fughe dalle configurazioni che disegnano. In effetti, agli inizi è facile che la danza cosmica tanto attesa sia invece una completa disintegrazione. È saggio fare pratica costruendo prima un sistema di due o tre stelle.

La struttura del programma CLUSTER è semplice: c'è un ciclo di inizializzazione seguito da un doppio ciclo. All'interno del doppio ciclo l'accelerazione, la velocità e la posizione di ogni stella vengono aggiornate in funzione della somma delle attrazioni delle altre stelle. Descriverò una versione particolarmente semplice del programma, in cui sono già incorporati l'incremento di tempo, la costante della forza e le masse stellari. Nonostante la sua semplicità, però, questa versione di CLUSTER sembra in grado di simulare quasi tutta la gamma dei comportamenti di un ammasso. Vengono usati tre insiemi di matrici. Il primo insieme segue le accelerazioni che le stelle subiscono lungo ciascuna delle tre coordinate ed è composto dalle matrici *ax*, *ay* e *az*. *ax(i)*, *ay(i)* e *az(i)* indicano le componenti *x*, *y* e *z* dell'accelerazione dell'*i*-esima stella. Non è necessario inizializzare il contenuto delle tre matrici all'avvio del programma. Il secondo insieme di matrici, *vx*, *vy* e *vz*, definisce le velocità: *vx(i)*, *vy(i)* e *vz(i)* registrano le componenti *x*, *y* e *z* della velocità dell'*i*-esima stella. Il terzo insieme di matrici registra le posizioni: *x(i)*, *y(i)* e *z(i)* sono le coordinate della posizione dell'*i*-esima stella. I valori di partenza per le matrici *x*, *y*, *z* e *vx*, *vy*, *vz* devono essere inizializzati all'avvio del programma.

Al segmento di inizializzazione segue il corpo principale del programma CLUSTER. Si può continuare indefinitamente a rientrare nel doppio ciclo, oppure il programmatore può stabilire condizioni specifiche che controllino il rientro. Il ciclo esterno prende in considerazione le stelle, una alla volta, e pone uguali a zero le componenti dell'accelerazione. Il ciclo interno, poi, calcola le forze esercitate su ogni stella dalle sue compagne dell'ammasso.

Assumiamo, per esempio, che la variabile di controllo del ciclo esterno sia *i* e quella del ciclo interno sia *j*. Il ciclo interno controlla prima di tutto se *i* è uguale a *j*. Se così è, il programma non richiede il calcolo delle forze: una stella non attrae se stessa. In ogni caso, calcolare la forza in questa circostanza porterebbe la macchina a cercare di dividere per zero. (È questa l'unica situazione che mi fa sentire davvero in pena per un calcolatore.) Quando *i* e *j* non sono uguali, CLUSTER usa la formula di Euclide per calcolare la distanza tra le stelle: le differenze tra le coordinate *x*, *y* e *z* sono elevate al quadrato e sommate l'una all'altra. Il risultato, naturalmente, è il quadrato della distanza. Il ciclo interno, poi, controlla se questo numero è uguale a 0. Se lo è, dovrebbe scattare qualche allarme perché il calcolatore sta per dividere per zero. La mia versione del programma avverte: «Collisione!».

Se nulla è andato storto, il ciclo interno calcola la distanza tra le stelle trovando la radice quadrata *d* della distanza quadrata trovata prima. Divide poi 1000 per il quadrato della distanza e determina così la forza. Il compito finale del ciclo interno è stabilire le componenti dell'accelerazione dell'*i*-esima stella, valore ottenuto sommando insieme i contributi di forza delle altre stelle. Per esempio, la componente *x* dell'accelerazione può essere scritta in modo generale come segue:

$$ax(i) \leftarrow ax(i) + f \times (x(j) - x(i)) / d$$

dove *f* e *d* rappresentano la forza e la distanza. Il rapporto tra la distanza *x* dall'*i*-esima alla *j*-esima stella e la distanza totale è proprio la frazione di forza che agisce sull'*i*-esima stella nella direzione *x*. Le componenti *y* e *z* dell'accelerazione

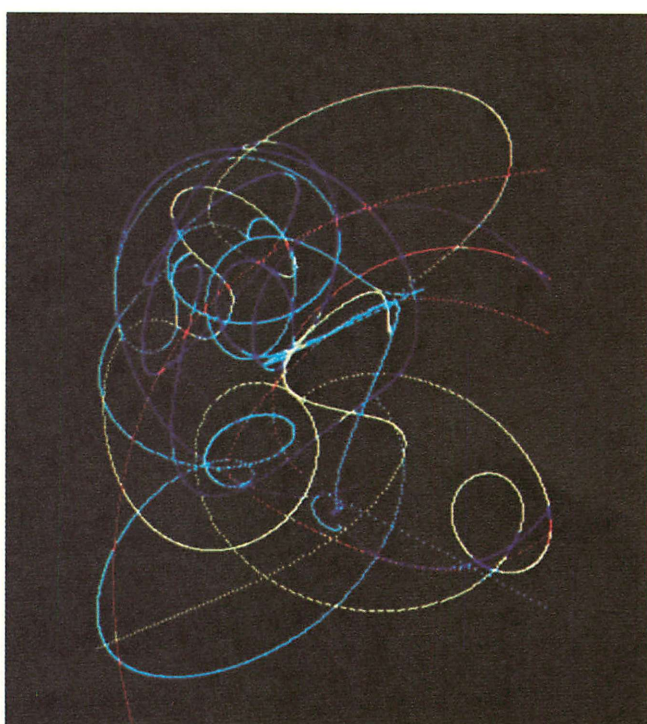
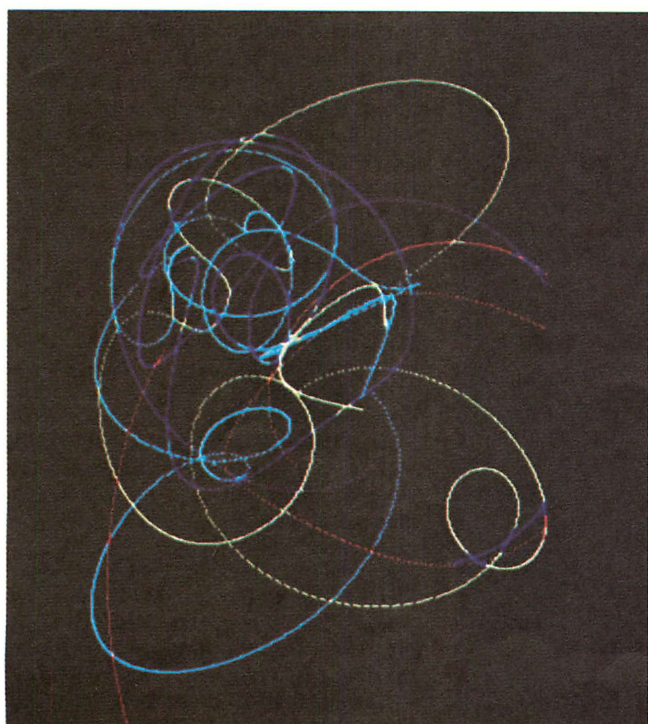
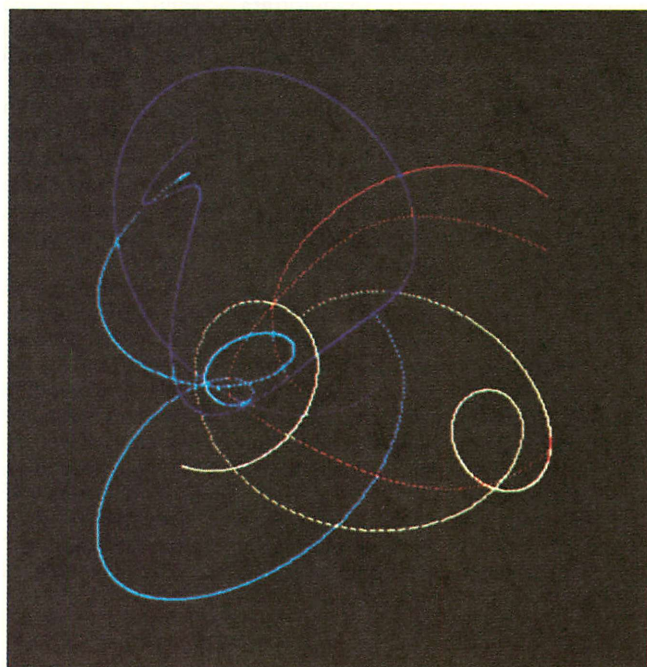
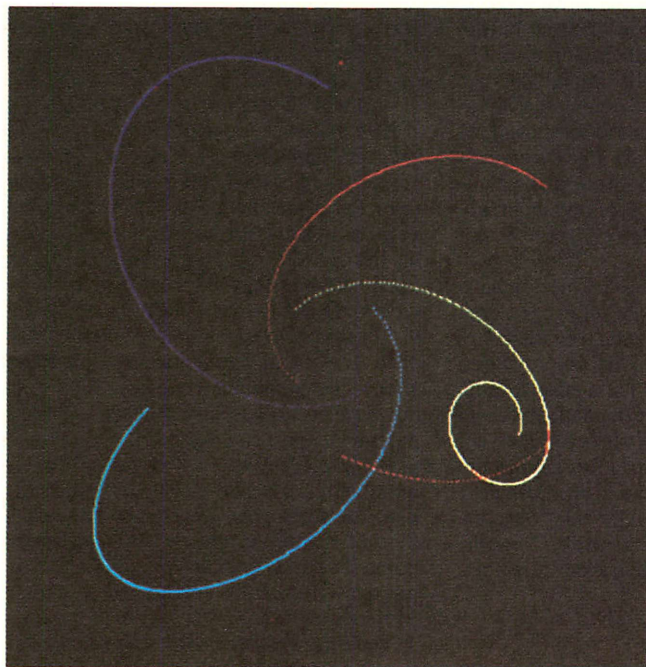
sono calcolate con formule analoghe. Altri due cicli, uno di seguito all'altro, completano il programma. Il primo aggiorna la velocità e il secondo la posizione. C'è qui una sottigliezza che mi è stata segnalata per la prima volta da John H. Hubbard, il matematico della Cornell University i cui consigli sono stati particolarmente utili per il calcolo dell'insieme di Mandelbrot (*si veda l'articolo a pagina 102*). È in effetti possibile calcolare la posizione prima di calcolare la velocità senza produrre strani risultati. I movimenti delle stelle, però, diventerebbero col tempo stranamente erronei, perché un'operazione di questo genere costituirebbe una violazione del principio della conservazione dell'energia.

Il ciclo che aggiorna la velocità non fa altro che sommare l'accelerazione alla velocità, secondo la formula:

$$vx(i) \leftarrow vx(i) + ax(i)$$

Si presuppone qui che l'incremento di tempo sia uguale all'unità di tempo in cui è espressa la velocità. Lo stesso tipo di formula è utilizzato per calcolare vy e vz. Altrettanto semplici sono i calcoli per la posizione effettuati nel ciclo finale:

$$x(i) \leftarrow x(i) + vx(i)$$

Gli elementi delle matrici y e z vengono aggiornati in modo analogo. Attingendo le informazioni dal ciclo finale, CLUSTER dispone ogni punto sulla superficie bidimensionale dello schermo, operazione compiuta usando le prime due coordinate della

Quattro stelle mettono in scena per qualche anno un balletto cosmico e poi lasciano il palcoscenico

posizione e sopprimendo la terza. Risultato naturale di questa disposizione è che z rappresenta la profondità; è facile immaginare di guardare nello spazio dietro lo schermo. I numeri prodotti dal programma di simulazione di ammassi sono a volte molto grandi e a volte molto piccoli. Per questo motivo è consigliabile utilizzare un'aritmetica in doppia precisione per evitare che i numeri rilevanti siano inavvertitamente arrotondati.

Il tempo impiegato da CLUSTER per terminare un ciclo di calcolo dipende dal numero di stelle del sistema. Con solo 10 stelle si ottiene un intrico esteticamente interessante; per produrre una complessità realistica sono necessarie 100 o addirittura 1000 stelle. Sfortunatamente, il numero dei passi del ciclo di calcolo fondamentale aumenta con il quadrato del numero di stelle dell'ammasso. Anche se i simulatori stellari hanno trovato un metodo semplice per aggirare questa particolare limitazione, sorgono altri problemi.

Il problema più grave nasce dal fatto che il programma è un sistema discreto che cerca di imitare un sistema continuo. Le orbite continue sono approssimate da una successione di salti che si allontanano sempre più dal percorso reale di una stella nell'ammasso. L'imprecisione potrebbe essere corretta in qualche misura dalla presenza di regolarità statistiche, ma negli incontri ravvicinati tra stelle il sistema amplifica in modo innaturale e disastroso l'effetto catapulta.

Per esempio, se il ciclo di calcolo pone una stella (Stella) vicino a un'altra stella (Astro), una potente spinta gravitazionale aumenta le componenti di accelerazione di entrambe le stelle. Questo effetto di amplificazione passa, attraverso il calcolo, fino alle componenti della velocità e di qui alle coordinate della posizione. Alla successiva iterazione Stella si trova già molto lontana da Astro e incapace di restituire il prestito gravitazionale. Si è creata una finzione di eccessiva energia cinetica. Gli ammassi artificiali afflitti da questo problema evaporano ancora più rapidamente di quelli reali. Ci sono due modi per sormontare la difficoltà; uno è arduo, l'altro è semplice. L'alternativa difficile richiede il calcolo di una orbita kepleriana per la coppia, orbita che viene mantenuta finché le due stelle si trovano in prossimità l'una dell'altra. I teorici prediligono questo metodo perché la formula orbitale è perfettamente precisa. Un modo facile, ma a volte impreciso, per trattare gli incontri ravvicinati consiste invece nel suddividere i «passi» temporali nel ciclo base del calcolo. Se lo vorranno, i lettori potranno aggiungere questo particolare procedimento alla versione avanzata di CLUSTER della quale ora fornirò una descrizione.

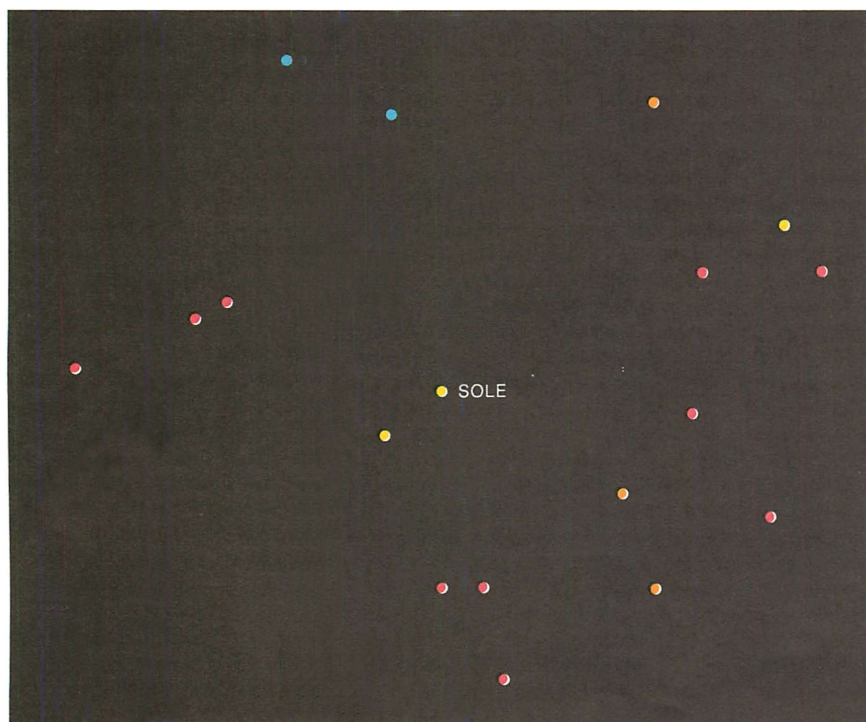
Con una serie di semplici modifiche si può derivare da CLUSTER un programma chiamato SUPERCLUSTER. Innanzitutto, SUPERCLUSTER include nel suo balletto stelle di masse differenti. La cosa è facilmente realizzabile immettendo le masse in una matrice *m*. Il calcolo della forza diventa un po' più complesso: la forza non è più proporzionale a $1/d^2$ bensì al prodotto delle masse diviso per d^2. SUPERCLUSTER, poi, tiene conto di alcuni tipi spettrali. Come nel caso della massa, all'inizio si deve riempire una matrice (chiamata *spec*), che però viene usata solo durante la fase di visualizzazione del ciclo base. I colori vanno dal blu per le stelle di tipo O al rosso per quelle di tipo M. Manca il verde. Il terzo miglioramento di CLUSTER consiste nel rendere possibili in entrambe le versioni del programma passi temporali arbitrari.

SUPERCLUSTER usa per il passo temporale una variabile chiamata *delta*. *Delta* deve essere specificata all'inizio e stabilisce la quantità di tempo simulato che intercorre tra cicli successivi. Naturalmente, questo elemento di tempo deve incidere sulle formule di aggiornamento per la velocità e la posizione: nelle formule per la velocità moltiplica l'accelerazione e nelle formule per la posizione moltiplica la velocità.

Si può ora descrivere il modo semplice per trattare gli incontri ravvicinati. Prima di tutto si deve dare una definizione di «ravvicinato». Si deve poi inserire nel programma, subito dopo il calcolo della distanza tra due stelle, un controllo di prossimità. Se sta avvenendo un incontro ravvicinato, SUPERCLUSTER sostituisce *delta* con un decimo del suo valore (almeno finché nessuna coppia di stelle si trova altrettanto vicina). Questo espediente aiuta certamente ad ammortizzare gli sbalzi improvvisi della gravità discreta, ma crea problemi ancora peggiori quando gli incontri sono davvero ravvicinati. Un incontro 10 volte più ravvicinato ora dà luogo a una forza gravitazionale 100 volte maggiore! Fortunatamente, incontri ravvicinati del tipo peggiore sono abbastanza rari. La tecnica di suddivisione del tempo è stata di impiego comune nei programmi di simulazione di ammassi tradizionalmente adottati dai professionisti.

Perché SUPERCLUSTER sia un programma significativo dal punto di vista astronomico, sono necessarie unità per la distanza, la massa e altri aspetti della realtà fisica. Una misura della distanza, comoda per i nostri fini, è l'unità astronomica (UA), uguale alla distanza media della Terra dal Sole. La massa può essere misurata in masse solari e la miglior misurazione per il tempo è in anni. Con queste convenzioni, la costante universale di gravitazione ha il valore approssimato di 39. SUPERCLUSTER usa questa costante (invece di 1000) nel calcolo della forza.

Ora è tutto pronto per mettere al lavoro i due programmi. Un esercizio preliminare per CLUSTER mette in gioco quattro stelle. Mettiamole agli angoli di un quadrato di pochi centimetri sullo schermo. È corretto dare un valore diverso da zero alla coordinata z, oltre che alle coordinate x e y ricordate in precedenza. Se il movimento è limitato al piano dello schermo, gli incontri ravvicinati sono molto più comuni. Le componenti di velocità dovrebbero essere piccole (sull'ordine di valori compresi tra -5 e $+5$) e dovrebbero specificare una direzione in senso orario, come se le quattro stelle fossero su una ruota.

SUPERCLUSTER può essere messo alla prova sul sistema illustrato in questa pagina, la regione galattica vicina alla Terra.

I nostri vicini galattici formerebbero un ammasso?

NOME DELLA STELLA	COORDINATE DELLA POSIZIONE			COORDINATE DELLA VELOCITÀ			COLORE	MASSA
	X	Y	Z	VX	VY	VZ		
STRUVE 2398	68	−365	631	−5,69	4,76	3,35	ROSSO	0,26
ROSS 248	464	−42	450	−8,75	1,13	−15,45	ROSSO	0,17
61 CYGNI	394	−377	433	−2,78	22,03	0,02	ARANCIONE	0,69
LALANDE 21185	−404	107	307	7,32	−0,47	−20,11	ROSSO	0,39
PROCYON 5	−295	658	68	2,38	0,75	−3,65	BLU	1,29
STELLA DI BARNARD	−7	−371	30	−0,87	24,20	16,78	ROSSO	0,21
EPSILON ERIDANI	408	534	−114	4,60	0,69	−0,50	ARANCIONE	0,74
WOLF 359	−462	136	62	−0,82	9,86	−5,94	ROSSO	0,10
SIRIUS	−98	514	−157	1,89	−2,21	−2,59	BLU	2,96
LUYTEN 726–8	487	219	−175	2,08	10,80	−0,41	ROSSO	0,19
ROSS 128	−683	44	13	2,51	−2,32	−4,09	ROSSO	0,21
SOLE	0	0	0	0,00	0,00	0,00	GIALLO	1,00
TAU CETI	646	307	−208	0,52	−6,62	3,92	GIALLO	0,85
ALPHA CENTAURI	−106	−86	−243	−1,95	4,68	4,51	GIALLO	1,03
LUYTEN 789–6	608	−235	−182	−6,75	10,81	10,56	ROSSO	0,13
LUYTEN 725–32	718	227	−233	4,70	6,16	0,51	ROSSO	0,21
ROSS 154	111	−536	−241	1,79	1,36	−0,11	ROSSO	0,24
EPSILON INDI	334	−194	−594	−3,54	17,71	2,28	ARANCIONE	0,69

Una tabella che elenca tutte le stelle, tranne tre, che si trovano nelle vicinanze del nostro sistema solare

Che cosa avverrebbe se il Sole e le stelle sue vicine fossero liberati dalla nostra galassia e fossero lasciati danzare senza fine nello spazio? Si formerebbe un ammasso? Abbia o meno rilevanza scientifica, è una domanda a cui è divertente rispondere. Inoltre, sono le uniche stelle di cui si conoscano con precisione la posizione e la velocità *(si veda l'illustrazione qui sopra).*

Gli ammassi di stelle sono aperti o globulari. Gli ammassi aperti sono formati da un migliaio di stelle, mentre gli ammassi globulari possono comprendere milioni di stelle. Finora, ricercatori come J. Garrett Jernigan, che lavora presso il Berkeley Space Sciences Laboratory dell'Università della California, sono stati in grado di trattare solo piccoli ammassi. Attualmente, non si riesce ad affrontare gli ammassi globulari. Pur con queste limitazioni, Jernigan e i suoi colleghi che hanno svolto in questo campo un lavoro pionieristico, come Sverre J. Aarseth dell'Università della California a Berkeley, osservano da decenni collassi di ammassi prodotti dal calcolatore. L'estensione del collasso è misurata considerando un volume sferico centrato all'interno dell'ammasso e contenente il 10 per cento della sua massa. Il raggio di questo volume è chiamato «raggio del 10 per cento». Il collasso sta per avvenire quando il raggio del 10 per cento diminuisce col passare del tempo. Inesorabilmente, il «nucleo» di un ammasso simulato diventa sempre più denso. Dato che le stelle simulate sono punti matematici, non accade nulla di terribile a questi ammassi. Non si forma alcun buco nero al centro. Questa, almeno, è stata l'esperienza dei teorici degli ammassi. Sembra, comunque, che ci siano poche prove di collassi estremi negli ammassi reali; qualcosa, là fuori, impedisce il collasso.

Sia gli esperimenti tradizionali, sia quelli moderni basati sulla simulazione, possono fornire una chiave di interpretazione. In varie occasioni, un piccolo numero di sistemi di stelle binarie al centro di un ammasso simulato ha virtualmente arrestato il collasso di regioni del nucleo. In uno degli esperimenti di Jernigan, il responsabile sembrava essere una singola binaria. Come è possibile? Secondo David Porter, uno studente di Jernigan, può essere che «binarie strettamente collegate sfreccino molto rapidamente una intorno all'altra e spingano energeticamente delle stelle vaganti intorno al nucleo o addirittura fuori fino a un insieme più slegato di stelle, chiamato alone, che circonda il nucleo. Questo potrebbe essere un meccanismo per impedire al nucleo di diventare troppo affollato».

Jernigan era un osservatore di stelle a raggi X. Una volta centrato lo studio sulla ricerca di sorgenti di raggi X negli ammassi, si è interessato sempre di più agli ammassi stessi come oggetti astronomici. La simulazione sembrava un modo efficace per studiarli.

Pur definendosi un novizio in materia, Jernigan ha dato un importante contributo al miglioramento dell'efficacia della simulazione. In CLUSTER e in programmi analoghi, un singolo ciclo di calcolo per n stelle richiede grosso modo n^2 passi. Il ciclo di Jernigan ne richiede solo $n \times \log(n)$. Jernigan organizza il suo ammasso raggruppando le stelle in coppie vicine. Ogni coppia è poi sostituita da una massa e da una velocità fittizie che riassumono il comportamento della coppia. Si applica poi lo stesso processo alle coppie come se fossero le stelle originali. Continuando in questo modo, si costruisce un insieme di nodi di massa più volte raggruppati in una struttura di dati detta albero. Il nodo singolo alla radice rappresenta simultaneamente tutte le stelle. Si possono poi calcolare i movimenti per il nodo centrale e per tutti i suoi rami fino alle singole stelle.

È questa la tecnica del futuro? Certamente, secondo Jernigan, aiuta a rendere le cose più veloci. È probabile, però, che le prossime generazioni di programmi per ammassi assomiglino di più alla varietà ibrida usata da Alan P. Lightman del Center for Astrophysics dello Harvard College Observatory e da Stephen L. W. McMillan dell'Università dell'Illinois a Urbana-Champaign: le stelle del nucleo sono trattate dai metodi di simulazione diretta descritti in precedenza; le stelle esterne al nucleo sono trattate invece con un modello di tipo statistico come se formassero un gas.

Per i lettori esperti nel linguaggio di programmazione APL c'è un'interessante nuova pubblicazione di Gregory J. Chaitin dell'IBM Thomas J. Watson Research Center di Yorktown Heights, New York. Si intitola *An APL2 Gallery of Mathematical Physics* ed è un libretto di 56 pagine che contiene la spiegazione di cinque importanti teorie fisiche, fra cui quelle che descrivono il movimento newtoniano e relativistico dei satelliti nello spazio. Vengono forniti listati in APL per programmi di calcolatore che illustrano ciascuna teoria.

Chaitin sarà lieto di inviare una copia del libro a ogni lettore che gli scriva al Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, N.Y. 10598.

Un sublime volo di fantasia su una base di dati deserta

di A. K. Dewdney

Le Scienze, settembre 1986

Studiai nervosamente il quadro degli strumenti del mio Cessna 182. Che pazzia mi aveva preso di tentare un volo solitario senza istruzioni? Forse avrei fatto meglio a farmi portare in volo da qualche esperto; ma fuori, nel nudo paesaggio di Meigs Field a Chicago, non si vedeva anima viva, nemmeno un meccanico a fumarsi una sigaretta al sole. Unico compagno nella folle missione, il manuale di volo che tenevo sulle ginocchia. Avevo passato soltanto dieci minuti a leggerlo.

Probabilmente mi sarei sentito molto più fiducioso con in mano un joystick o una manopola. Da bambino avevo visto almeno dieci volte *Hellcats of the Navy* e conoscevo già i movimenti fondamentali. Ma questo aereo era chiaramente molto più raffinato: per controllarlo avevo a disposizione solo una tastiera davanti a me.

Con un sospiro andai a pagina 42 del manuale: il tasto F2 per dare un po' di gas, F4 per dare tutto gas. Temerario sì, ma non troppo, provai a schiacciare F2. In sottofondo, il tranquillo ronzio del motore salì di un semitono. La pista cominciò a corrermi incontro e improvvisamente ebbi un momento di panico. L'aereo iniziò quasi subito a uscire dalla striscia di cemento. Nella mia ignoranza, avevo trascurato di leggere qualcosa a proposito del timone e della ruota anteriore. La mia unica salvezza erano gli alettoni, i tasti 4 (sinistra) e 6 (destra). Con mia grande sorpresa, uno di essi mi riportò lentamente sulla pista.

Davanti a me, a sole cento yarde, si intravvedevano le profonde acque azzurre del lago Michigan. Più gas. F2 F2 F2 F2. Ancora di più. Il motore rombava a pieno regime, ma l'aereo si rifiutava di sollevarsi. La pista correva sempre più veloce sotto di me e fui travolto da un'altra ondata di panico. Quale magico comando ci voleva per andare su? La mia vita mi scorse davanti agli occhi come un lampo. Scene dell'infanzia: volavo su un aeroplanino controllato solo da un paio di flap sulla coda. Il timone di profondità! Febbrilmente, cercai «timone di profondità» sul manuale. Il timone di profondità controlla l'angolo d'attacco del velivolo: sollevandolo si alza il muso dell'aereo. Schiacciai il tasto 2 appena in tempo per evitare un tuffo mortale nell'acqua. Lentamente, la severa geometria di terra e acqua cominciò ad allontanarsi. Ero in volo! L'aereo sembrava salire per conto suo. La mia visuale sulla costa occidentale del lago Michigan cresceva costantemente e volai per un po' verso nord, ipnotizzato dal mio successo.

Come molti lettori avranno subito capito, la carlinga era lo studio di casa mia e i comandi erano sulla tastiera del mio calcolatore personale, su cui girava un programma per la simulazione di volo. L'ambiente familiare, però, non diminuiva affatto il senso dell'avventura; il fatto che sia l'aereo sia il paesaggio fossero semplici simulazioni serviva solo a espandere la mia immaginazione. Si possono fare cose che nessuno si azzarderebbe a tentare su un velivolo vero. Anche il conto dei danni sarebbe immaginario. Virai verso sud-ovest, puntando verso Chicago. Volevo calare come un falco su Madison Avenue e mettere nello scompiglio il traffico con la mia audacia. Secondo l'orologio posto sul pannello degli strumenti erano le 4 e 30, l'inizio dell'ora di punta pomeridiana.

Mentre mi avvicinavo a Chicago, ebbi la terribile sensazione che mancasse qualcosa. Sul lungolago si vedevano solo pochi edifici e le strade erano completamente prive di traffico. Non c'era proprio nessuno in giro. Era scoppiata una guerra nucleare? Mi accorsi di aver preso il grosso altimetro per un orologio, ma anche la scoperta che era molto più presto delle 4 e 30 non poteva spiegare lo spopolamento della grande metropoli. Con riluttanza, dovetti concludere che nel programma per la simulazione del volo non c'era la gente. La sua base di dati era già riempita dai paesaggi nordamericani, per non parlare dei particolari geografici di quattro aree metropolitane e di oltre 20 aeroporti.

In voli più recenti, la mancanza di compagnia ha finito col preoccuparmi sempre meno: è già troppo divertente volare. Molte altre persone sembrano pensarla allo stesso modo. Il programma si chiama FLIGHT SIMULATOR ed è stato per molti mesi in testa o quasi alle classifiche di vendita del software ricreativo. Molti altri programmi per la simulazione di volo sono disponibili sul mercato. Solo dietro a FLIGHT SIMULATOR, però, c'è, insieme con un grande numero di altri esperti nella produzione di forti emozioni, Bruce A. Artwick, un ingegnere del software di Champaign, Illinois. Forse più di chiunque altro, Artwick è responsabile del decollo dei programmi per la simulazione di volo che possono girare su calcolatori domestici.

La simulazione di volo è stata per lungo tempo dominio quasi esclusivo di piloti in addestramento; tuttavia grazie ad Artwick e ad alcuni altri programmatori è ora virtualmente di proprietà comune.

Che cosa deve saper fare un programma per la simulazione di volo, perché l'utente raggiunga l'illusione di trovarsi in una situazione reale? Innanzitutto deve creare e ri-creare un paesaggio da un particolare punto di vista varie volte al secondo. Mentre il velivolo simulato percorre i suoi cieli simulati, il punto di vista cambia e l'aspetto del paesaggio deve essere ricalcolato. Inoltre, per ogni punto di vista ci deve essere un appropriato angolo di vista determinato simulando la fisica di un aereo in volo. Per esempio, durante una virata del velivolo si devono modificare in modo coordinato sia l'angolo di vista sia il punto di vista. Il ciclo base di calcolo utilizza una grande quantità di trucchi e di compromessi già di per sé affascinanti.

La base di dati geografica è fondamentale per FLIGHT SIMULATOR ed è costituita da un'ampia lista degli oggetti necessari per formare i territori su cui si vuole volare (o su cui si vuole evitare di precipitare). Si ottiene la scena del proprio volo tracciando una gerarchia «dall'alto verso il basso» di informazioni. Un blocco di memoria contrassegnato U.S.A. contiene i nomi di regioni quali Chicago, New York e Seattle. La posizione del velivolo è data da una terna ordinata di coordinate orientate rispetto al parabrezza: x, la sua latitudine, y, la sua altezza sul livello del mare, e z, la sua longitudine. FLIGHT SIMULATOR confronta poi le coordinate con i limiti definiti per ogni regione del blocco U.S.A. Per esempio, se x e y specificano un punto dell'area di Chicago, il programma esegue un'operazione chiamata caricamento della base di dati: la lista che descrive l'area di Chicago viene caricata dal disco nella memoria del calcolatore.

Quando puntai per la prima volta con l'aereo verso Chicago, non era visibile alcun edificio. Dopo un po', però, divenne improvvisamente visibile la Sears Tower; poco dopo, si precisarono i dettagli. Questa è la strategia di FLIGHT SIMULATOR per rimanere entro i limiti di velocità e di memoria di un microcalcolatore: una caratteristica del paesaggio appare soltanto quando è realmente necessaria. Due particolari sottoprogrammi di verifica determinano la comparsa o la scomparsa di una caratteristica. Il primo, *ifin2d*, controlla solo due coordinate dell'aereo, la latitudine e la longitudine, alle quali, come ho già detto, associa una regione geografica.

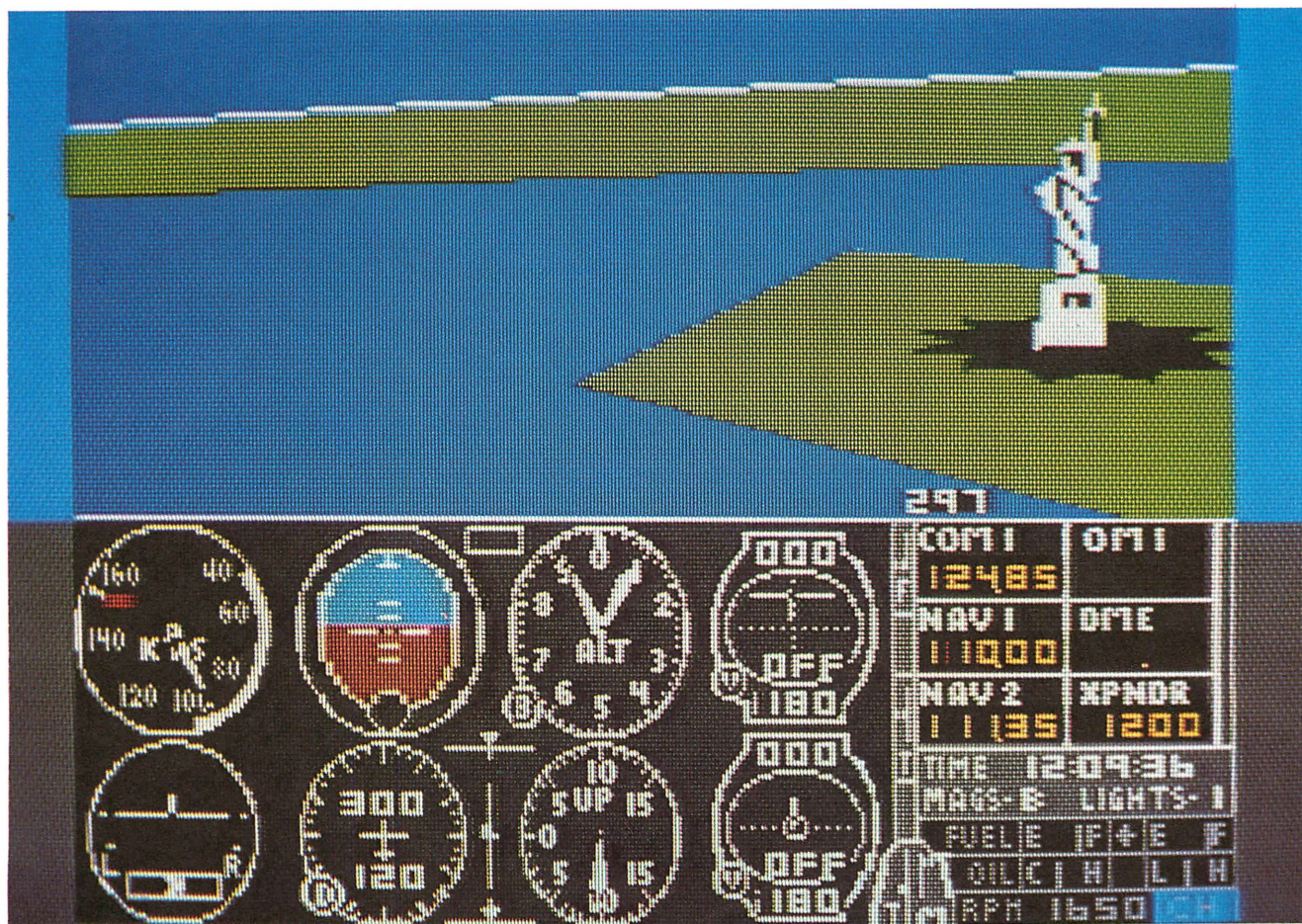
Il secondo sottoprogramma di verifica, *ifin3d*, stabilisce se una data caratteristica del paesaggio circostante è abbastanza vicina da giustificarne la visibilità. Mentre mi avvicinavo a Chicago, *ifin3d* continuava a ricalcolare la mia distanza tridi-

mensionale da certi punti di riferimento della città sottostante. Quando mi avvicinai a 12 800 metri dalla Sears Tower, una replica in miniatura dell'edificio spuntò dal paesaggio come una confusa pila di pixel in distanza. Arrivato a una distanza di 7680 metri, divennero improvvisamente visibili maggiori dettagli. Per ogni nuovo insieme di coordinate, FLIGHT SIMULATOR passa in rassegna con la lista di visualizzazione tutte le caratteristiche del paesaggio in memoria e le controlla secondo le regole di visualizzazione di *ifin3d*. Se un oggetto supera la verifica, il programma lo disegna così come appare dall'attuale punto di vista. In caso contrario non viene disegnato e il programma passa al successivo oggetto della lista.

In generale, ogni oggetto o caratteristica dello scenario simulato è definito da una lista di punti, linee e superfici. Dato che l'aspetto di ciò che si vede viene calcolato cinque volte al secondo, abbastanza velocemente per sfruttare la persistenza delle immagini sulla retina, le linee e le superfici devono essere disegnate nel più breve tempo possibile. Anche le più elementari operazioni grafiche del programma, quindi, sono state progettate con grande cura. Artwick le ha descritte nell'affascinante volume *Microcomputer Displays, Graphics, and Animation.*

Per esempio, il metodo fondamentale per disegnare una linea evita moltiplicazioni e divisioni perché su un microcalcolatore richiedono tempi da 10 a 100 volte superiori rispetto alle somme e alle sottrazioni. Il metodo si basa invece su un'accurata analisi del dispendio computazionale di ogni istruzione. Il brillante algoritmo per disegnare una linea che ne risulta è formato in sostanza da sole quattro istruzioni: una somma, una verifica, una ramificazione e un'istruzione grafica. Supponiamo che sia appena stato tracciato il punto (a, b) su una linea con pendenza (positiva) pari a 1/4. Il programma entra in un ciclo in cui ripetutamente somma 1 alla coordinata a, aggiunge la pendenza 1/4 a una speciale somma chiamata errore di tracciato e poi controlla l'errore stesso. Quando quest'ultimo è maggiore o uguale a un certo intero positivo, il programma esce dal ciclo e traccia il punto sucessivo, in questo caso ($a + 4, b + 1$).

FLIGHT SIMULATOR disegna superfici in un modo che sembra contorto quando lo si descrive, ma che è estremamente veloce da eseguire. In breve, il programma disegna il poligono che racchiude la superficie e poi ne riempie l'interno. Quando è stata calcolata la forma del poligono, il sottoprogramma grafico identifica le righe di scansione del video che attraversano il poligono. Per ognuna di queste righe di scansione, i lati del poligono che vengono attraversati sono identificati e poi ordinati secondo l'ordine in cui la riga di scansione li incrocia, da sinistra verso destra. Il sottoprogramma prende poi in considerazione ogni pixel, o elemento di immagine, della riga di scansione in ordine sinistra-destra. Quando la riga incontra il primo lato del poligono, incomincia a visualizzare punti di colore adeguato alla superficie da rappresentare, fino a quando incontra il lato successivo. Arrivata al lato ancora successivo, incomincia da capo.

La tecnica di disegno appena descritta è basata su un fondamentale risultato della topologia piana: qualsiasi poligono divide il piano in due parti, una interna e una esterna. Di conseguenza, qualsiasi linea che attraversi un lato del poligono passa dall'interno (la superficie) all'esterno (non-superficie) o viceversa. La maggior parte delle superfici disegnate da FLIGHT SIMULATOR è delimitata da poligoni abbastanza semplici. Poco dopo la mia partenza da Meigs Field, per esempio, la sponda del lago Michigan si estendeva da sotto l'aereo fino a un lontano orizzonte seguendo un'insenatura poligonale parzialmente chiusa da una striscia di terra orientata verso nord. Le linee di scansione dell'acqua azzurra della baia erano interrotte dal-

***La Statua della Libertà vista dall'abitacolo in* FLIGHT SIMULATOR**

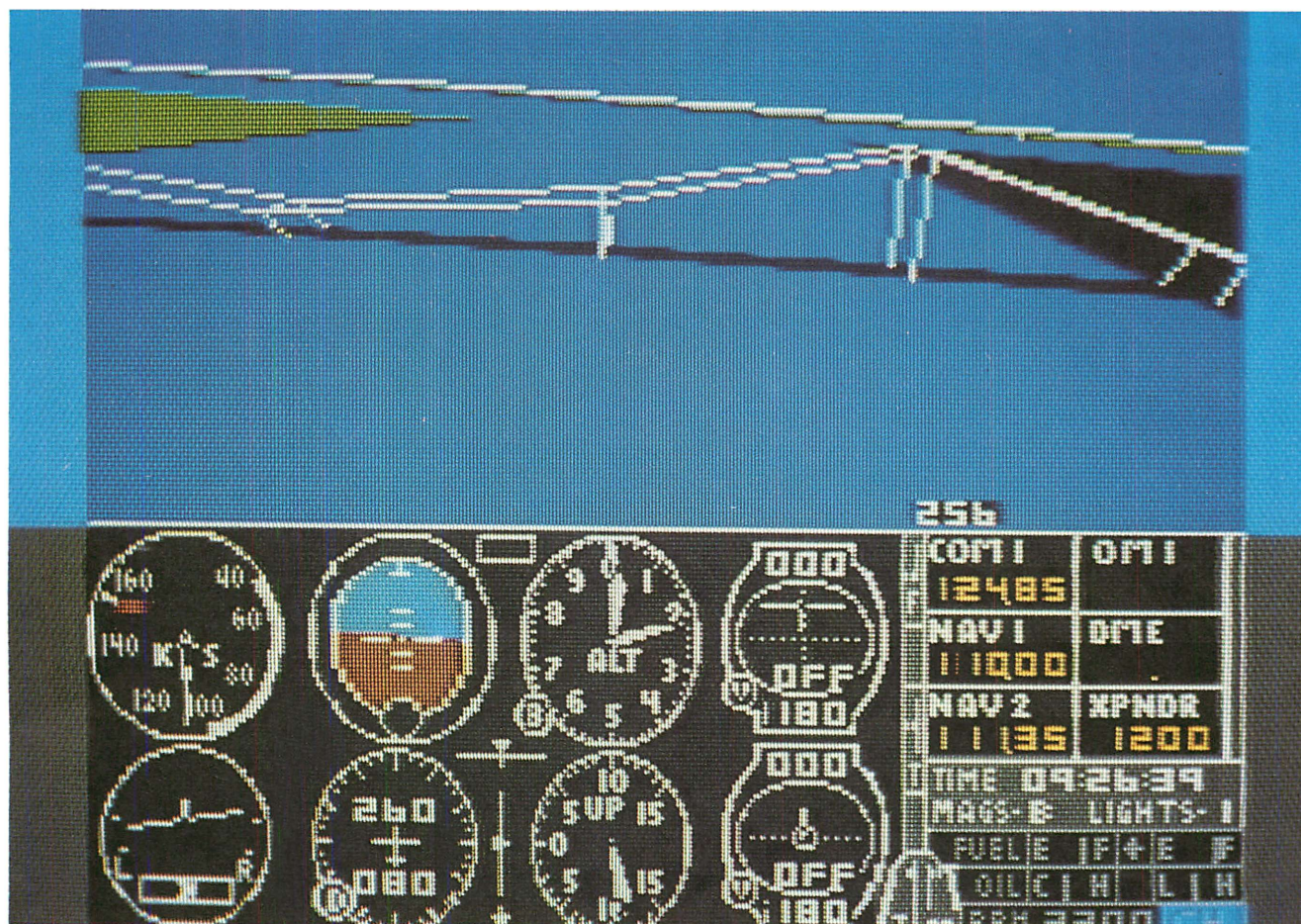
la striscia di terra e poi continuavano a riempire il corpo centrale del lago.

Gli estensori di programmi di grafica si sono spesso trovati a riflettere sul cosiddetto problema delle linee nascoste: in che modo un programma può disegnare un oggetto tridimensionale opaco perché sia visibile solo il lato più vicino all'osservatore? Alcuni metodi richiedono la soluzione di lunghe equazioni, un carico di calcolo che FLIGHT SIMULATOR non può permettersi. La soluzione di Artwick, applicabile su larga scala solo a paesaggi semplici, consiste nel disegnare l'intero oggetto in modo da lasciare per ultimo il lato più vicino all'osservatore. Le parti dell'oggetto che non risultano visibili dal punto di vista dell'osservatore vengono semplicemente cancellate da quelle che lo sono. La rappresentazione delle sottili torri del World Trade Center di New York è un buon esempio di questa tecnica brutale, ma efficace. Da ogni angolo, i lati più vicini di ciascun edificio oscurano i più lontani e la torre più vicina nasconde la più lontana.

Dopo la scoperta dello spopolamento di Chicago, decisi di fare nuovamente rotta su Meigs Field. Il vero proprietario dell'aereo era forse in ansia per la sua sorte. Eppure, c'era ancora molto che volevo imparare. Da una parte, mi sentivo intralciato dal fatto di conoscere solo alcuni dei comandi. Dall'altra dovevo ancora tentare un'acrobazia veramente drammatica. Decisi di spingermi sul lago Michigan per fare un tentativo. Quelli degli alettoni erano i miei comandi preferiti. Schiacciando il tasto dell'alettone di sinistra feci virare l'aereo a sinistra. Davanti a me l'orizzonte si piegò verso destra e lo scenario sembrò scivolare dolcemente lungo la pendenza. In accordo con le leggi dell'aerodinamica, il muso del Cessna girò verso sinistra. Volando nuovamente dritto e in orizzontale, battei qualche volta il tasto 8 e sentii quasi fisicamente cadere il muso dell'aereo. Fuori del finestrino, l'orizzonte e lo scenario salirono nella parte alta della mia visuale: l'aereo stava scendendo in picchiata. L'indicatore di velocità superò le 150 miglia orarie. Tentai di azionare i comandi del timone. Il tasto della somma spostò un poco a destra il timone e l'aereo cominciò pericolosamente a ruotare. Riportai al centro il timone e contemplai il paesaggio salirmi incontro con ingannevole lentezza. Era tempo di effettuare la mia acrobazia.

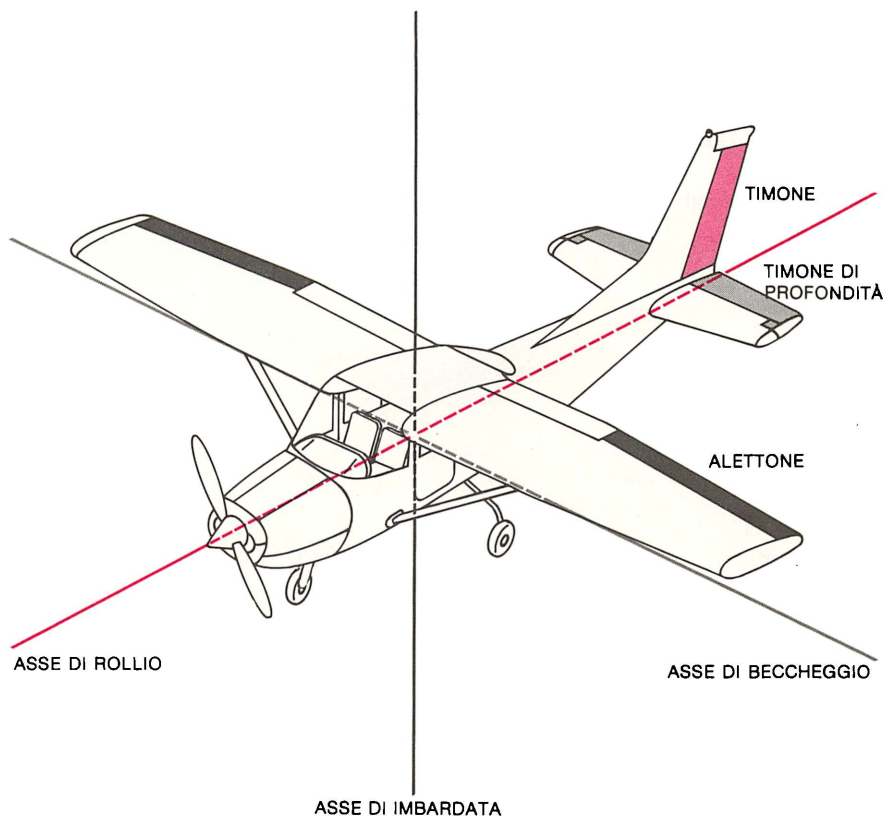
I piloti tentano un giro della morte molto meno spesso dei programmatori. Sapevo istintivamente, però, che mi serviva un bel po' di velocità per la manovra. Prima giù, poi su e poi ancora più su. Schiacciare dei tasti non era certo come tirare con forza la cloche; comunque battei diligentemente il tasto 2. L'aereo uscì rombando dalla picchiata e poi salì verso l'empireo delle linee azzurre. Vidi l'orizzonte tuffarsi sotto il finestrino e attesi. Non avevo la più pallida idea di che cosa fosse successo. Tutto era silenzioso. Forse ero finito in stallo ed ero precipitato al suolo? I miei strumenti non mi erano di alcun aiuto; ne conoscevo al massimo metà, e anche quelli molto vagamente.

Improvvisamente comparve nel finestrino un paesaggio rovesciato, verde terra sopra e azzurro cielo sotto. Ero riuscito nella manovra? Ero troppo confuso per riuscire a rispondermi. Ora avevo solo terra davanti a me. Poi l'orizzonte salì al suo livello giusto. Cercai di abbassare il muso per evitare un altro giro della morte. L'orizzonte ondeggiò, poi si stabilizzò. C'ero riuscito. Era ormai chiaro che sono una di quelle persone dotate che riescono a volare quasi senza istruzioni. Davanti a me apparve Meigs Field.

La pista principale si estendeva da sud a nord. Il mio piano di volo, dalla posizione a sud-est del campo d'atterraggio in cui mi trovavo, prevedeva di volare verso ovest e poi girare a nord una volta allineato con la pista. Qualcosa, però, andò storto. Non riuscii ad allinearmi esattamente e l'avvicinamento alla pista avvenne da

Una sfida alla morte sotto il ponte di Manhattan simulato

un'altezza eccessiva. L'incubo peggiorava sempre più. Abbassai il muso dell'aereo per perdere quota rapidamente, lottando nel contempo per guadagnare qualche metro verso est. Ma la pista abbandonò lo schermo e persi il mio orizzonte. Si fece sentire un segnale acustico d'avvertimento mentre la terra mi balzava incontro. Accidenti! Mi ero schiantato. Il finestrino si riempì di crepe e in mezzo apparve la scritta «Crash». Ero morto? Sarebbe stato un vero peccato se la base di dati di Chicago avesse perso il suo unico abitante vivo.

Per simulare queste avventure, FLIGHT SIMULATOR deve continuare a ricalcolare l'aspetto del paesaggio così come viene visto dall'abitacolo del Cessna. I comandi di un aereo possono venir raggruppati secondo i tre tipi di rotazione che essi provocano (*si veda l'illustrazione di questa pagina*). Gli alettoni ruotano l'aereo lungo una linea che va dal muso alla coda, l'asse di rollio. I timoni di profondità provocano una rotazione attorno a una linea parallela alle ali, l'asse di beccheggio. Il timone tende a ruotare l'aereo attorno al suo asse di imbardata, una linea verticale rispetto al pilota. Se la posizione dell'aereo in un certo istante è data dalle coordinate x, y e z, come fa il programma a stabilire dove si troverà una frazione di secondo dopo?

La direzione del muso del Cessna e la velocità nel momento preso in considerazione costituiscono un vettore, e il vettore dà origine a una nuova posizione. Il programma calcola una nuova direzione per l'aereo - e di conseguenza un nuovo angolo di vista - a partire dalle equazioni di volo. Per ogni ciclo di visualizzazione, gli ingressi delle equazioni comprendono i valori attuali dei comandi e di condizioni ambientali quali pressione e densità atmosferica, velocità del vento e temperatura. Le equazioni di volo incorporano le caratteristiche di volo dell'aereo e applicano le leggi della fisica ai valori in ingresso. Si aggiungono le forze di spinta, di gravità, di portanza e di resistenza, e si considerano gli effetti inerziali della massa e della geometria dell'aereo. La soluzione delle equazioni è la nuova direzione di volo. Le equazioni sono troppo complesse per essere risolte rapidamente e Artwick le ha sostituite con tabelle di valori. Per esempio, una tabella elenca i valori di portanza dell'aereo per ogni angolo di attacco; gli angoli sono dati per ogni incremento di un decimo di grado. Le tabelle hanno un vantaggio in più: le caratteristiche del volo possono essere modificate semplicemente cambiando le tabelle senza definire nuove formule.

Tre tipi di superfici di controllo per la rotazione di un velivolo

I lettori che vogliano lanciarsi in un «progetto pilota verticale» possono acquistare un programma per la simulazione di volo in vendita per i più popolari fra i modelli di calcolatori domestici. Artwick ha guidato lo sviluppo del programma più famoso, venduto dalla Microsoft Corporation di Bellevue, Washington. Ha avuto anche un ruolo importante nello sviluppo di un programma chiamato FLIGHT SIMULATOR II, messo in commercio dalla Sublogic Corporation di Champaign, Illinois. Entrambi i programmi sono disponibili per i calcolatori personali Apple II, IBM, Commodore 64 e Atari. Di recente è stata annunciata una terza produzione di Artwick per il calcolatore MacIntosh.

Dopo qualche volo di prova, forse il lettore vorrà tentare qualcuna delle avventure da far rizzare i capelli proposte in un nuovo libro di Charles Gulick, intitolato *40 Great Flight Simulator Adventures*. Sono grato a David Wehlau di Waltham, Massachusetts, per aver richiamato la mia attenzione su questo libro. Ogni avventura inizia quando l'utente fa partire uno dei programmi di simulazione di volo citati in precedenza e batte i dati necessari tratti dal libro, che comprendono latitudine, longitudine, altezza, posizione dei comandi, ora e così via.

Istantaneamente vi potete trovare a poche centinaia di metri dal ponte di Manhattan, su una rotta che vi porta a passare direttamente sotto il ponte. Oppure, in un'avventura intitolata «Atterraggio d'emergenza su San Clemente», il motore è andato in avaria mentre volavate a 5000 piedi sull'isola di San Clemente, al largo della costa della California. Siete capaci di atterrare sull'isola in volo planato? In una terza avventura chiamata «Zona del crepuscolo» siete parcheggiati su una pista erbosa. L'orologio posto sul pannello degli strumenti vi dice che da tre minuti è passata mezzanotte, ma fuori c'è luce. Il velivolo, fino a dove è visibile dal finestrino laterale e da quello posteriore, è tutto nero. Il decollo è lento e, una volta in volo, non si vede niente né dai finestrini né con il radar. Che cosa bisogna fare? Personalmente preferisco una base di dati deserta.

Dove si introduce il lettore ai piaceri del calcolo

di Brian Hayes

Le Scienze, dicembre 1983

Calculemus
-GOTTFRIED WILHELM VON LEIBNIZ

Alcuni dei migliori giochi con il microcalcolatore sono ora dotati di una funzione chiamata TBIC. Le lettere stanno per *The boss is coming* (arriva il capo) e quando si preme il tasto assegnato alla funzione lo schermo diventa immediatamente scuro e silenzioso. Sono qui rappresentati, mi sembra, i due poli della risposta pubblica alla recente proliferazione di calcolatori a basso costo. Da una parte il calcolatore è una macchina per il mondo degli affari, uno strumento capitalistico; dall'altra è un mezzo di intrattenimento così frivolo da dover essere nascosto, come i fumetti.

Non intendo sminuire alcuna delle due applicazioni pratiche dei calcolatori: gli affari e l'industria o quel genere di giochi il cui scopo principale è mettere alla prova i riflessi del giocatore. Il calcolo a fini utilitaristici è indubbiamente importante e, in quanto ai videogiochi, la loro costruzione può essere tra le espressioni più elevate dell'arte della programmazione. Va osservato, tuttavia, che né l'uno né l'altro di questi impieghi del calcolatore implicano in profondità la questione di ciò che un calcolatore è e di ciò che esso è in grado di fare.

Esiste un vasto territorio compreso tra la programmazione per affari e i videogiochi, tra VisiCalc e Space Invaders. In questo territorio sono comprese le applicazioni del calcolatore a tutte le arti e scienze e, forse più ovviamente, alla matematica. È compreso, l'uso del calcolatore per simulare aspetti del mondo naturale e delle società umane. Sono compresi, inoltre, molti impieghi che, propriamente parlando, non sono affatto «usi» del calcolatore, ma servono piuttosto a centrare l'attenzione sul calcolatore stesso e sulla natura del calcolo meccanizzato.

Anche gli strumenti dell'uomo d'affari si possono a volte applicare a problemi relativi alla teoria e alla pratica del calcolo meccanico.

Considererò nel nostro caso alcune questioni sollevate da applicazioni non convenzionali dei programmi chiamati *spreadsheet*, vale a dire tabelloni o fogli elettronici.

Un tabellone di carta è un grande foglio diviso in righe e colonne che potrebbe essere impiegato per analizzare il bilancio di un'azienda. A ogni reparto potrebbe essere attribuita una colonna, e una riga a ogni conto di ricavo o di perdita. In altre colonne e righe potrebbero essere inseriti i totali e le percentuali per ogni reparto o per ogni conto.

Il tabellone elettronico riproduce questa struttura sullo schermo di un tubo catodico, ma con alcune rilevanti differenze. Sulla carta, una casella (definita come l'intersezione di una colonna e una riga) può portare o una dicitura, come il nome di un reparto, o un numero. In un tabellone elettronico a una casella può essere assegnata anche una formula matematica. Così la casella alla fine di una riga potrebbe contenere una formula che richiede di sommare i valori immessi in tutte le altre caselle della riga. Sullo schermo appare il numero risultante dalla valutazione della formula in questo caso il totale, ma il contenuto di fondo della casella è la formula stessa e non il numero. Se si modifica uno degli ingressi della riga, il totale viene automaticamente ricalcolato.

Il primo dei programmi di tabellone elettronico fu VisiCalc, elaborato nel 1978 da Daniel Bricklin, allora studente alla Harvard University School of Business, da Robert Frankston e Dan Fylstra. Si dice che ne siano state vendute più copie di qualsiasi altro programma per calcolatore. In seguito sono state introdotte decine di altri programmi che funzionano su principi analoghi e lo stesso VisiCalc è stato più volte revisionato. La maggior parte degli esperimenti qui descritti è stata effettuata con due più recenti programmi di tabellone: 1-2-3, ideato da Mitchell Kapor della Lotus Development Corporation di Cambridge, Massachusetts, e Multiplan, un prodotto della Microsoft Corporation di Bellevue, Washington. Nella maggior parte dei casi andrebbero altrettanto bene altri programmi di tabellone.

Sebbene concepito per l'analisi finanziaria, il tabellone elettronico è in grado di fare molto di più. È una matrice bidimensionale di caselle in cui il valore di ogni casella può essere fatto dipendere da qualsiasi altra casella o gruppo di caselle. È sorprendente vedere quanto della struttura matematica del mondo possa essere inserito in tale formato; il tabellone elettronico risulta un contesto di grande generalità per la descrizione di relazioni logiche e matematiche.

Un semplice esempio può contribuire a una migliore comprensione di come si utilizza un tabellone e di quali siano le sue potenzialità. Ogni casella è identificata dalle sue coordinate; nella maggior parte dei programmi le colonne sono individuate da lettere e le righe da numeri, partendo dall'angolo a sinistra in alto. Supponiamo che tanto alla casella A1 quanto alla casella A2 sia assegnato un valore numerico 1. Si immette poi una formula nella casella A3: il valore della posizione A3 è dato dalla somma del valore della casella immediatamente al di sopra di essa e del valore della casella sopra quest'ultima. In altre parole, A3 è uguale al contenuto di A2 più il contenuto di A1 e assume il valore 2.

Quello che è stato compiuto finora è banale: si tratta di uno schema molto elaborato per esprimere la relazione $1 + 1 = 2$. Ora, però, è possibile riprodurre la formula di A3 in molte altre caselle. (L'esatta procedura per questa riproduzione varia da programma a programma, ma tutti i programmi prevedono questa possibilità.) Supponiamo che la formula di A3 sia copiata nelle caselle da A4 ad A10. Ciascuna di quelle caselle conterrà un valore uguale alla somma dei valori delle due caselle superiori. Si noti che le formule sono tutte identiche ma, dato che vengono applicate a valori diversi, i risultati non lo sono. I numeri che compaiono, leggendo dall'alto verso il basso, sono 1, 1, 2, 3, 5, 8, 13, 21, 34 e 55.

Vi sono molti modi per generare la serie di Fibonacci con un calcolatore e la maggior parte di essi fa un uso molto più efficiente delle risorse della macchina. C'è però qualche cosa di caratteristico nella strategia del tabellone: non è algoritmica. In quasi tutti i linguaggi di programmazione un compito o la soluzione di un problema sono definiti in termini di un algoritmo, cioè di una serie di istruzioni esplicite che debbono essere eseguite in una sequenza ben determinata. Un algoritmo è un po' come una ricetta: potrebbe iniziare con «Mescolare farina, lievito e acqua, lasciar lievitare e infine cuocere». Se effettuassimo le stesse operazioni in un altro ordine avremmo un risultato molto differente. Il tabellone non ha questa caratteristica di ordinamento temporale. Nelle caselle non va inserita una sequenza di passi che porta dal problema alla soluzione, ma una struttura statica che cerca di racchiudere tutta insieme l'intera procedura. È una descrizione, anziché una ricetta: afferma che il pane è fatto di farina, lievito e acqua mescolati, fatti lievitare e cotti.

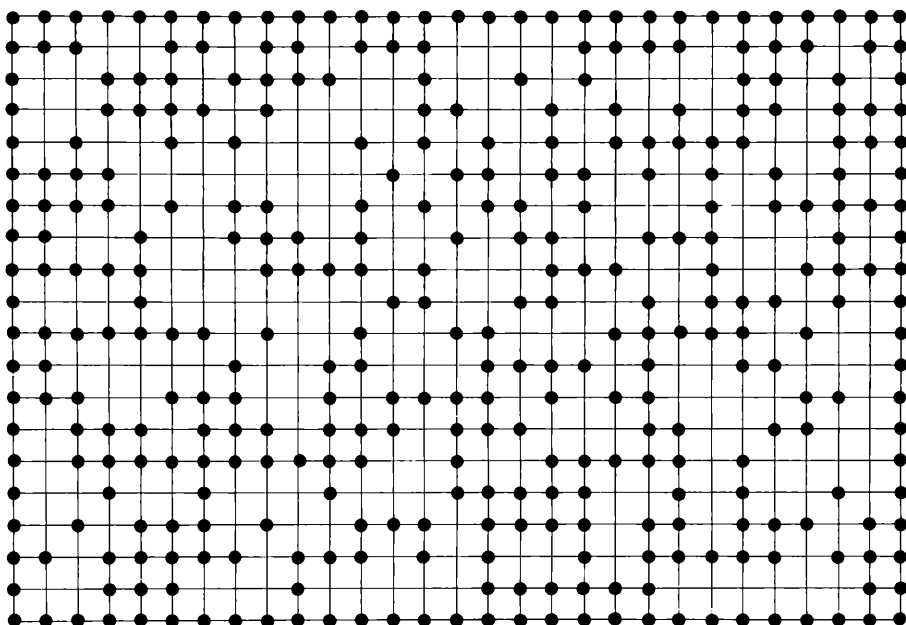
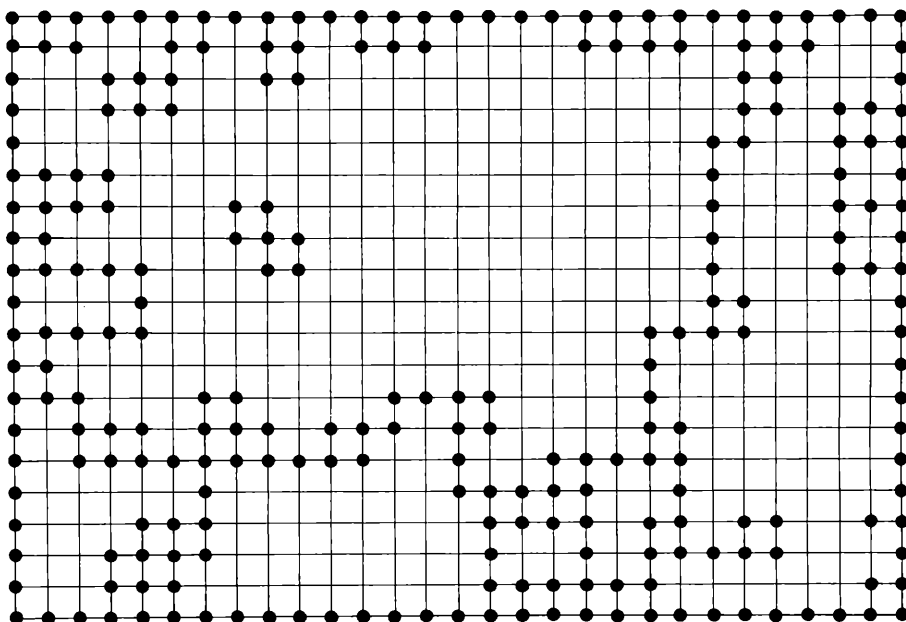
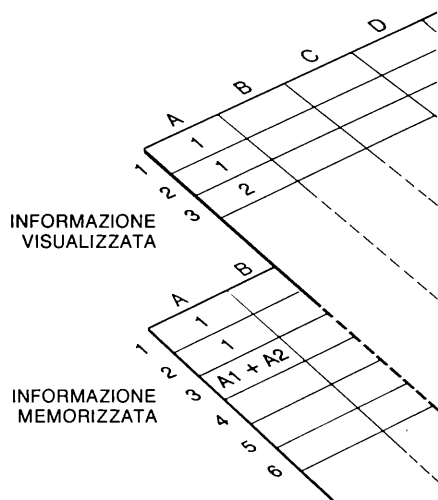
Possiamo chiarire meglio la distinzione tra un algoritmo e una descrizione statica con un altro esempio. Consideriamo il procedimento per moltiplicare due matrici di numeri, ciascuna delle quali abbia tre colonne e tre righe. L'algoritmo standard inizia con le istruzioni di moltiplicare ogni elemento della prima colonna della prima matrice per ogni elemento della prima colonna della seconda matrice, di sommare i tre risultati e immagazzinare la somma come primo elemento della matrice prodotto. Le stesse istruzioni sono poi ripetute per le altre otto combinazioni di righe e colonne. In un tabellone il problema è posto in altra forma e può sfruttare l'analogia matematica tra una matrice matematica e una schiera di caselle. Invece di scrivere una sequenza di istruzioni, si definisce semplicemente la matrice prodotto, ponendo ogni casella uguale a una formula che rappresenta l'opportuna combinazione di colonne e righe. Una volta immesse le formule, queste vengono risolte «tutte in un colpo» e appare l'intera matrice prodotto.

A un livello più profondo, naturalmente, un calcolatore che lavora sotto la direzione di un programma di tabellone sta in realtà eseguendo un algoritmo. Un calcolatore che abbia una sola unità centrale di elaborazione può fare una cosa sola alla volta e quindi le caselle sono valutate secondo una certa sequenza. Chi usa il programma, però, di solito non tiene conto della sequenza e in effetti spesso ne è del tutto all'oscuro: non ha quindi bisogno di pensare in termini di algoritmi.

Non è certo mia intenzione suggerire che la modalità di pensiero non algoritmica sia in qualche modo migliore di quella algoritmica: qualcuno può anche preferirla, ma è in gran parte questione di gusto. Quando una procedura diviene molto complessa, ci sono buone ragioni per raccomandare l'algoritmo, più facilmente divisibile in pezzi maneggevoli. Per risolvere il problema tutto in una volta bisogna capirlo tutto in una volta. Sembra comunque probabile che ci siano certi problemi o classi di problemi che si prestano naturalmente a una formulazione non algoritmica.

Un campo in cui la disposizione bidimensionale del tabellone risulta appropriata è la costruzione di sistemi di automi cellulari. Lo studio di questi sistemi venne iniziato negli anni cinquanta da John von Neumann e Stanislaw Ulam, che a quel tempo si interessavano soprattutto di modelli autoriproducentesi. Le regole fissate da von Neumann e Ulam richiedono uno «spazio cellulare uniforme» in cui ogni cellula rappresenta un automa, cioè una macchina che ha solo un numero finito di stati possibili. Lo spazio è uniforme nel senso che le leggi che governano lo stato degli automi sono le stesse per tutte le cellule. Un'ulteriore limitazione è che lo stato di una cellula può essere influenzato solo dalla sua storia e da quella dei suoi immediati vicini.

Le condizioni che definiscono un sistema di automi cellulari possono essere facilmente soddisfatte da un programma di tabellone. In linea di principio, il numero di stati disponibili a una cellula (che sarà rappresentata da una casella del tabellone) è enorme (forse 10^{100}), ma è chiaramente finito e può essere ridotto a un piccolo numero se sembra opportuno; per esempio, a una cellula può essere assegnata una formula che può assumere solo due possibili valori, quali 0 e 1. La richiesta di uniformità aggiunge un'interessante limitazione: essa implica che ogni cellula in cui è scritta una formula contiene sempre precisamente la stessa formula. (C'è più di un modo per decidere se due formule sono uguali. Supponiamo che una formula nella casella A1 si riferisca alla casella B1 immediatamente sotto di essa. Una formula in A2 dovrebbe essere considerata identica se anch'essa si riferisce a B1, con identità di «indirizzo assoluto», oppure se si riferisce a B2, con mantenimento della relazione geometrica. La seconda interpretazione è di solito più utile e sembra più consona alle idee di von Neumann e Ulam, ma entrambi gli schemi sono accettabili se applicati in modo coerente. I sottoprogrammi del tabellone per copiare i contenuti delle caselle forniscono un semplice test operativo di uniformità. Una schiera di caselle può essere considerata uniforme se, una volta inserita in una casella, una formula può poi essere copiata dal programma in tutte le altre.)

Von Neumann riuscì a dimostrare che

Reticolo generato da un programma di tabellone elettronico che simula la percolazione

«Assottigliando» il reticolo si rendono più evidenti i percorsi continui

Struttura di un tabellone elettronico

esiste una configurazione autoreplicantesi di cellule e ci riuscì seguendo una via estremamente complessa, mostrando che esiste un costruttore universale che può creare qualsiasi modello e deve essere quindi in grado di costruire il proprio stesso modello. La dimostrazione richiede 200 000 cellule con 29 stati possibili. Per quanto ne so io, il modello non è mai stato costruito realmente, né manualmente né con l'aiuto di un calcolatore. È comunque pensabile che rientri nelle possibilità dei programmi di tabellone più potenti.

Un sistema molto più semplice in grado di autoreplicarsi fu escogitato nel 1960 da Edward Fredkin, del Massachusetts Institute of Technology. Ogni cellula ha solo due stati possibili, viva e morta, che possono essere rappresentati dai numeri 1 e 0. Lo stato di una cellula della generazione successiva è determinato dallo stato presente dei suoi quattro adiacenti ortogonali, cioè delle quattro cellule immediatamente a nord, est, sud e ovest. Se il numero dei vicini vivi è pari (0, 2 o 4), la casella muore o rimane morta. Invece, una casella con un numero dispari di vicini vivi (1 o 3) vive.

È semplicissimo esprimere questa regola in una formula per tabellone, soprattutto con programmi che includono una funzione per l'aritmetica modulare. Nel caso della casella B2 la formula è semplicemente (B1 + C2 + B3 + A2) modulo 2. L'effetto della formula è sommare i valori delle quattro caselle adiacenti, dividere per 2 e conservare solo il resto, che è necessariamente 0 o 1. Non rimane poi che copiare la formula (in modo che i riferimenti delle caselle preservino le stesse relazioni geometriche) in tutte le caselle di una regione del tabellone. In realtà c'è un'altra sottigliezza nella costruzione del sistema: sono necessarie due copie dello spazio cellulare. Una copia rappresenta l'attuale generazione e una preserva lo stato della generazione precedente. Lo stato presente di una cellula si fonda sul numero degli adiacenti vivi nella generazione precedente.

Quando il tabellone è disposto secondo le regole di Fredkin e viene fornito un modello iniziale, ogni ciclo di ricalcolo produce un nuovo modello. Dopo alcuni cicli, appaiono quattro copie della configurazione originale. In seguito le stesse copie sono copiate quattro volte e il modello iniziale è stato quindi riprodotto 16 volte. Il numero di cicli necessario per la riproduzione dipende dalla complessità del modello iniziale; nel caso più semplice (una sola casella viva) i quattro discendenti appaiono immediatamente.

Può risultare affascinante osservare il progredire di una colonia in crescita. La simmetria si mantiene tutte le volte e alcuni dei modelli hanno una sorprendente forma stellata. C'è un ritmo nel processo: il perimetro dell'area occupata si espande in modo continuo, ma l'interno si riempie periodicamente di un denso strato di cellule e poi si svuota di nuovo.

Sicuramente il più noto fra i sistemi di automi cellulari è il gioco «Vita», inventato da John Horton Conway dell'Università di Cambridge e fatto conoscere al mondo da Martin Gardner.

Il gioco, proprio come i suoi automi in continua moltiplicazione, si è ormai esteso a quasi tutti i sistemi di calcolatori e linguaggi di programmazione. C'è una buona ragione: ricompensa largamente l'attenzione prestatagli, sia essa un'attenzione da spettatore oppure un'analisi approfondita.

Nel gioco Vita di Conway, le regole non sono definite per assicurare la replica di un modello ma per massimizzare la varietà o minimizzare la prevedibilità. Anche in questo caso ogni cellula ha due stati possibili, ma il suo stato è influenzato non più da quattro, ma dalle otto cellule più vicine, comprese quelle che hanno solo un vertice in comune con la cellula di partenza. Se una cellula è viva, continuerà a vivere solo se ha due o tre vicini vivi. Se ne ha di meno si dice che muore di solitudine e se ne ha di più muore per sovraffollamento. Per una cellula non viva è possibile nascere solo se ha esattamente tre vicini vivi.

Una specificazione algoritmica di questo procedimento tende a essere molto ripetitiva: bisogna esaminare una data cellula, contare i suoi vicini, decidere se deve vivere o morire, poi passare a una cellula successiva e poi a una successiva ancora finché tutte sono state verificate. Di solito la ripetizione in un programma si ottiene con quella particolare struttura chiamata *loop* o ciclo, che in questo caso viene eseguita una volta per ogni cellula. Quando si codifica il gioco Vita in un tabellone, la ripetizione c'è ancora ma è spaziale anziché cronologica: si immette la stessa formula in ogni casella di un'ampia configurazione.

Vi sono molti modi di scrivere una formula per valutare lo stato di una cellula nel gioco Vita. Il migliore che io abbia visto (nel senso di quello che procede più velocemente) è stato escogitato da Ezra Gottheil della Lotus. Il procedimento base consiste nel moltiplicare il valore della cellula in esame per 9, ottenendo un risultato pari a 0 o 9, e poi sommare i valori delle otto cellule circostanti. Il risultato è confrontato con una tabellina che dà il nuovo stato della cellula per tutti i possibili valori della somma (vale a dire i valori compresi tra 0 e 17).

Idealmente Vita andrebbe giocato su una matrice cellulare di estensione infinita. Una delle attrattive del gioco è che certe piccole configurazioni iniziali si sviluppano, dopo appena poche generazioni, in magnifici fiori simmetrici; altre configurazioni emettono proiettili compatti che scivolano via a distanze infinite. L'evoluzione di un organismo si modifica, quando esso supera i confini del mondo. Una matrice infinita è in ogni modo impossibile e quando si lavora con un tabellone i limiti pratici sono, in realtà, piuttosto stretti e dipendono dalle possibilità del programma stesso, dalla capacità di memoria del calcolatore e dalla propria pazienza. (Il tempo necessario per creare una nuova generazione è, grosso modo, proporzionale al numero di cellule coinvolte.) Una strategia per creare una matrice che non abbia confini, pur essendo su un'area finita, è quella di definire adiacenti le cellule disposte lungo bordi opposti; si modifica così la topologia del tabellone. Unendo due bordi in tal modo si genera un cilindro o, se al foglio si fa compiere una torsione, un nastro di Möbius. L'unione dei quattro bordi a due a due dà luogo a un toro.

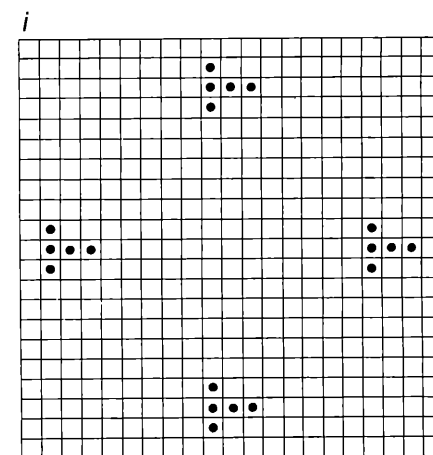
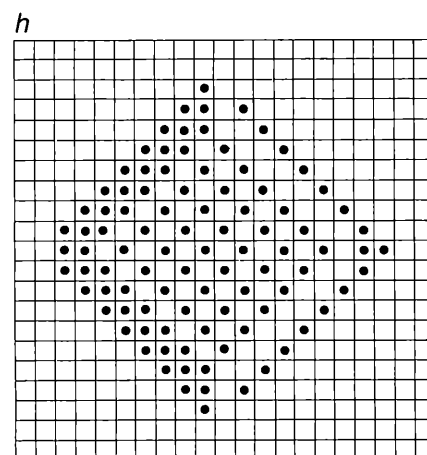
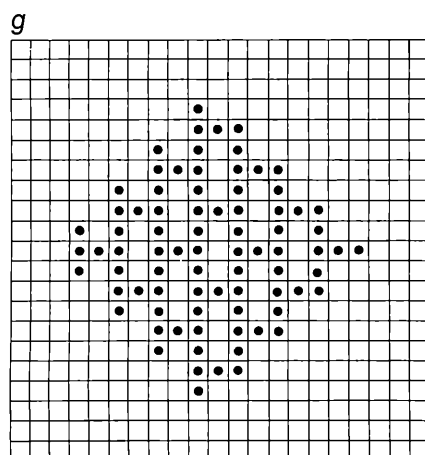
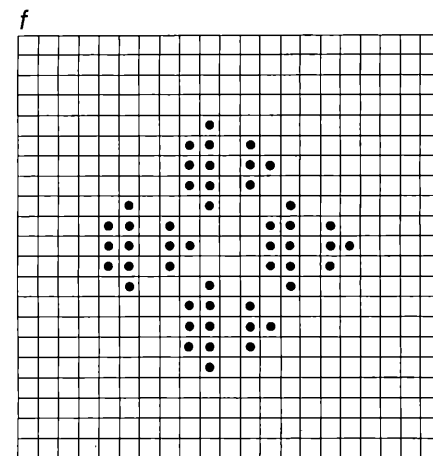
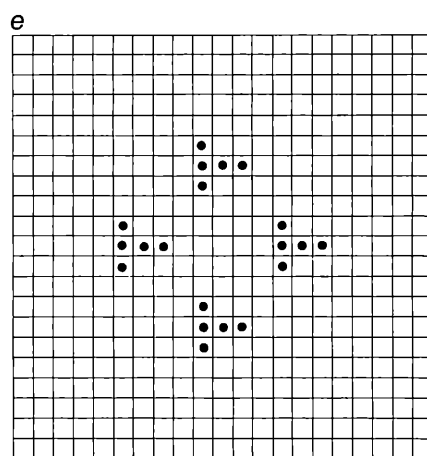
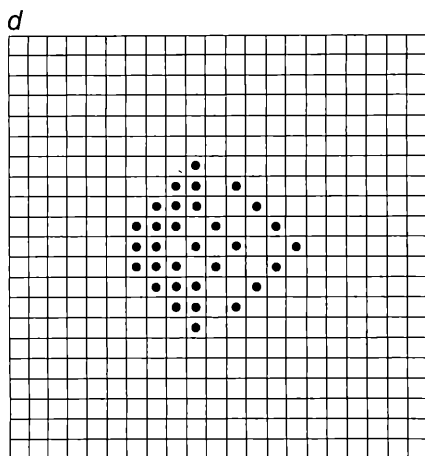
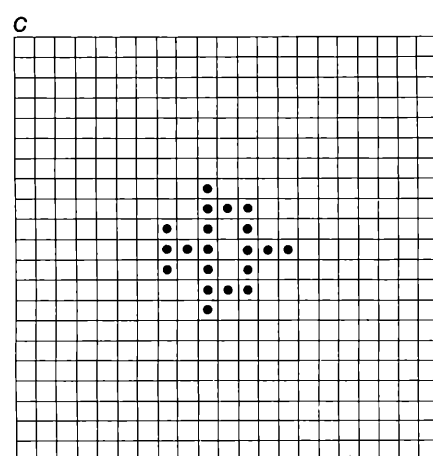
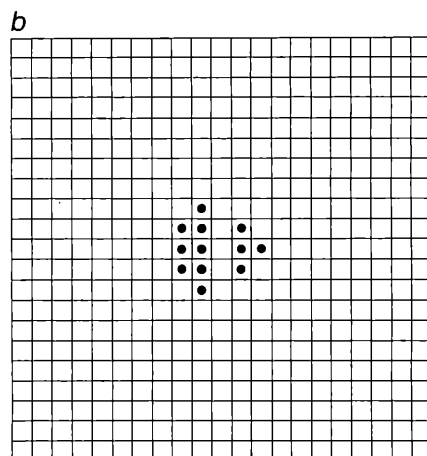
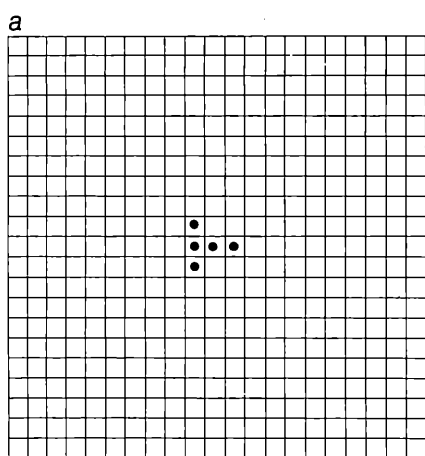
Il modello di Ising è la simulazione di un sistema fisico che, almeno superficialmente, assomiglia a certi automi cellulari, benché la sua interpretazione sia molto differente. Il modello, che è stato introdotto dai fisici tedeschi Wilhelm Lenz ed Ernest Ising negli anni venti, può rappresentare molti fenomeni fisici, ma viene per lo più usato per la descrizione dei materiali ferromagnetici. Ogni posizione in un reticolo rappresenta il momento angolare di spin e quindi il momento magnetico di un atomo. Ogni spin ha modulo fisso, ma l'asse di spin può essere orientato «verso l'alto» o «verso il basso». Quando tutti gli spin hanno la medesima orientazione, il materiale è completamente magnetizzato; quando invece la loro orientazione è casuale, la magnetizzazione è nulla.

La realizzazione del modello di Ising con un tabellone elettronico è un po' più complicata di quella dell'automa replicantesi di Fredkin o del gioco Vita di Conway. Anche qui lo stato di un dato spin è influenzato dalle caselle più vicine, e precisamente dalle quattro ortogonali. Nel modello di Ising, tuttavia, c'è un elemento di casualità che rappresenta l'effetto della temperatura diversa da zero. Se gli spin delle caselle vicine sono tutti orientati verso l'alto, anche lo spin di quella data casella avrà la tendenza a orientarsi verso l'alto, ma non è sicuro che ciò avvenga; la probabilità è inversamente proporzionale alla temperatura.

Alcuni esperimenti fatti da me con un modello di Ising su tabellone elettronico

hanno dato risultati discordanti. Le proprietà del modello bidimensionale sono note esattamente dal 1944, quando Lars Onsager della Yale University ha risolto il sistema in modo analitico (anziché con una simulazione numerica). Quando la temperatura viene abbassata fino a un valore limite (la temperatura di Curie), gli spin cominciano a fluttuare selvaggiamente e il materiale si magnetizza completamente. Non ho assistito a tale transizione di fase, ma la cosa non mi sorprende. Il modello di Ising richiede macchine con enormi potenze di calcolo. Per ottenere risultati accurati è necessario un reticolo molto grande ed è necessario esaminare tutte le possibili configurazioni degli spin, il che può richiedere molte ore anche con un programma efficiente e un elaboratore molto veloce. All'Istituto di fisica teorica dell'Università della California a Santa Barbara è stato costruito un calcolatore apposito per il modello di Ising che calcola 25 milioni di spin al secondo. Con il modello realizzato su tabellone elettronico, gli spin esaminati al secondo sono 25.

Anche se gli eventi interessanti che si verificano attorno alla temperatura di Curie non si possono osservare, il modello di Ising su tabellone sembra simulare bene altre proprietà dei materiali magnetici. A temperatura elevata la disposizione degli spin non evidenzia alcuna struttura, come ci si aspetterebbe. A bassa temperatura la magnetizzazione del reticolo è ovvia. Si creano spontaneamente gruppi irregolari di spin allineati, mentre gruppi di polarità opposta sembrano scontrarsi ai loro confini. Sorprendente, almeno per me, è stata la comparsa di una fase antiferromagnetica in cui tutti gli spin hanno orientazione alternata (uno verso l'alto, uno verso il basso ecc.). Allo zero assoluto la fase antiferromagnetica sembra la configurazione più stabile, ma ciò potrebbe riflettere un mio errore nella costruzione del modello.

Un reticolo piano consente di trattare

Sistema di automi cellulari ideati da Edward Fredkin in cui ciascuna struttura riproduce se stessa

Versione su tabellone del modello bidimensionale di Ising del ferromagnetismo

anche altri sistemi fisici. Un fenomeno che mi interessa è la percolazione, che descrive non solo il modo in cui si fa il caffè, ma anche la struttura di alcuni polimeri, la conducibilità delle leghe, l'efficienza delle reti telefoniche, la propagazione degli incendi nei boschi e la diffusione delle malattie infettive. In un semplice modello della percolazione le caselle del reticolo potrebbero rappresentare possibili posizioni di atomi di rame in una matrice isolante. In prima approssimazione, la probabilità che una posizione sia occupata è proporzionale alla concentrazione di rame. Ciò che interessa è la probabilità di formare una catena ininterrotta di atomi di rame in qualche dominio del reticolo; è questa probabilità che determina la conducibilità del materiale.

Anche un modello della percolazione su tabellone elettronico comporta qualche forma di casualità. Il sistema più semplice consiste nel servirsi di uno spazio cellulare uniforme, in cui tutte le posizioni hanno la stessa probabilità di essere occupate e sono tutte indipendenti fra loro. Il risultato è una disposizione casuale di caselle libere e occupate. È allora necessario determinare se esiste o meno un percorso continuo attraverso lo spazio. Una soluzione corretta sarebbe quella di eseguire una ricerca sistematica di ogni percorso potenziale, ma non mi è chiaro come si possa fare senza ricorrere ad algoritmi. Una tecnica più brutale, ma ugualmente utile, è quella di includere nella formula per ogni casella una funzione che elimini ogni atomo con meno di due atomi adiacenti. Dato che ogni atomo che fa parte di una catena deve avere almeno due atomi adiacenti, la catena non viene modificata da questa procedura, ma ogni volta che il tabellone viene ricalcolato si eliminano le strade senza uscita e i gruppi rimasti isolati.

Un programma di tabellone elettronico non è certamente il mezzo ideale per rappresentare ognuno di questi modelli matematici o fisici. Per un lavoro serio, ciascun modello andrebbe realizzato con un programma «dedicato». Nel caso del gioco Vita ho messo a confronto varie versioni su tabellone con un programma (un algoritmo!) scritto nel linguaggio nativo di un microelaboratore. Per configurazioni della stessa dimensione, il programma in linguaggio macchina è quasi 100 volte più veloce del tabellone più veloce. La ragione non è difficile da trovare: anche se una casella non può mai avere un valore diverso da 0 o 1, il programma di tabellone calcola il valore fino alla quindicesima cifra decimale.

Se non può vantare la dote dell'efficienza, il tabellone ha in compenso la virtù della versatilità. Scrivere un programma in linguaggio macchina per giocare a Vita è ben più che uno svago pomeridiano. Inoltre, il programma non può fare nient'altro, mentre la semplice matrice di caselle collegate in un tabellone costituisce un mezzo per la soluzione di problemi di impressionante generalità. Le possibilità dei programmi vanno ben al di là di quan-

to delineato in precedenza. Risulta evidente che si può generare qualsiasi serie di numeri i cui termini siano definiti da funzioni algebriche o trigonometriche. Si può costruire un crivello per individuare i numeri primi a partire da una breve formula ripetuta centinaia di volte. Si può rappresentare un campo fisico prendendo l'indirizzo di ciascuna casella come le sue coordinate in uno spazio bidimensionale. Come dice la pubblicità: «L'unico limite è la vostra immaginazione».

È vero? Una matrice di formule interdipendenti può servire a calcolare tutto ciò che è calcolabile? Il meccanismo è non solo generale ma universale? La domanda ha già avuto una risposta, per il caso di una matrice infinita. Conway ha dimostrato che il mondo cellulare del gioco Vita ha risorse sufficienti per costruire una macchina di Turing, il modello concettuale di un calcolatore universale. Dato che lo si potrebbe usare per giocare a Vita, si potrebbe impiegare un tabellone infinito anche per creare la macchina di Turing.

Tale risultato è sicuramente degno di nota, ma, anche se si può trascurare il requisito di avere un'area infinita, la dimostrazione non ha nessun significato pratico: la vita è troppo breve e il gioco Vita troppo lungo. Io trovo più promettente una impostazione meno formale per stabilire la portata dei programmi di tabellone. È il metodo («o la va o la spacca») di applicare i programmi a vari problemi ed esercizi scelti nel repertorio classico della scienza del calcolatore. I casi interessanti sono verosimilmente quelli con una soluzione algoritmica molto efficiente. Ne è un esempio la Torre di Hanoi, in cui parecchi dischi sono impilati in ordine decrescente su uno dei tre perni; scopo del gioco è spostare i dischi uno per volta, evitando di mettere un disco più grande sopra uno più piccolo, finché non risultino impilati nello stesso ordine su un altro perno. La soluzione standard si basa su un algoritmo ricorsivo che stabilisce esplicitamente lo stadio finale della procedura e poi si basa su di esso per definire gli stadi precedenti.

Si può risolvere la Torre di Hanoi con metodi completamente non algoritmici? Si può fare con un tabellone? Arrivare a tale soluzione non costituirebbe sicuramente una dimostrazione che un tabellone possa fare tutto quello che può fare un algoritmo, ma ne estenderebbe considerevolmente il campo d'azione. Si noti che esiste un metodo di soluzione banale che non è ammesso. Si può risolvere manualmente il gioco, annotando la configurazione dei dischi a ogni stadio e poi scrivere una serie di formule che specifichino la transizione da una configurazione all'altra. È caratteristico di tali metodi forzati che, non appena si opera un piccolo cambiamento nelle condizioni iniziali, quale può essere rappresentato dall'aggiunta di un ulteriore disco, si renda necessario ricominciare. In ogni caso, una soluzione corretta dovrebbe rimanere valida per raggruppamenti di qualsiasi entità senza cambiamenti o perlomeno soltanto con cambiamenti nelle dimensioni della matrice. Si noti anche che almeno un programma di tabellone, 1-2-3 della Lotus, comprende un semplice linguaggio algoritmico. Ovviamente, anche questo strumento deve essere bandito.

Un altro caso interessante è il problema delle otto regine, in cui si tratta di disporre otto regine su una scacchiera standard in modo che nessuna regina sia sotto attacco. In questo caso il «formato» del problema - la matrice finita delle caselle - è allettante. Non c'è nessuna difficoltà a rappresentare una scacchiera con un programma di tabellone, ed è facile anche scrivere una formula che indichi se una casella è sotto attacco di una regina posta in qualsiasi punto della scacchiera. (La formula si limita a verificare se c'è un valore diverso da zero lungo tutte le righe, le colonne e le diagonali per una distanza di otto caselle.) Se fosse tutto qui, tuttavia, il problema non avrebbe attirato l'attenzione di Carl Friedrich Gauss, che lo esaminò nel 1850, ma non riuscì a risolverlo. Sembra che per ogni casella sia necessaria l'informazione non solo sull'attuale configurazione della scacchiera, ma anche sulle configurazioni precedentemente sperimentate. La difficoltà di fornire un'informazione come questa in una rappresentazione statica del problema fa pensare che gli algoritmi abbiano un futuro sicuro.

Dove si parla dell'automa finito: un modello minimale delle trappole per topi, dei ribosomi e dell'anima umana

di Brian Hayes

Le Scienze, febbraio 1984

I calcolatori più potenti non hanno né hardware né software: sono fatti di puro pensiero. La più celebre tra queste macchine astratte è quella ideata nel 1936 dal matematico inglese Alan Mathison Turing. È una macchina in grado di fare più di quanto abbia mai potuto fare un calcolatore con componenti in silicio: in realtà può calcolare tutto ciò che è possibile calcolare. Esiste poi una classe di calcolatori concettuali che, pur non raggiungendo l'onnipotenza della macchina di Turing, sono altrettanto interessanti. Si tratta delle cosiddette macchine finite, o automi finiti, che determinano le caratteristiche minime di un calcolatore digitale funzionante.

Una corretta definizione di macchina finita richiederebbe un grado di rigore matematico inadeguato per questa sede. Si può però chiarire la natura del concetto con qualche esempio. Cercando qualche macchina finita, ne ho trovato un ottimo esempio in una stazione della metropolitana sulla Lexington Avenue di New York. È un cancelletto girevole di vecchio tipo, fatto non con il consueto treppiede compatto d'acciaio, ma di quattro bracci incrociati di quercia, consumati da innumerevoli mani e fianchi.

Il cancelletto ha due stati: bloccato e sbloccato. Supponiamo che si trovi nello stato bloccato, in modo che non si possano ruotare i bracci. Inserendo un gettone si ottiene una certa modificazione del meccanismo interno che consente ai bracci di muoversi; in altre parole, il gettone induce una transizione allo stato sbloccato. Ruotando i bracci di 90 gradi si provoca un'altra transizione che riporta il cancelletto allo stato di blocco. Nella figura della pagina a fronte si vedono le transizioni rappresentate in modo schematico. Gli stati del sistema sono rappresentati da nodi (riquadri) e le transizioni da archi (frecce) che li collegano.

Nell'analisi finita del cancelletto, l'inserzione di un gettone e la rotazione dei bracci sono i possibili input del sistema. La risposta della macchina dipende sia dall'input sia dallo stato al momento dell'input. Spingere i bracci quando il cancelletto non ha ricevuto un gettone non vi garantirà un viaggio in metropolitana. Inserire un gettone quando i bracci sono già sbloccati è altrettanto inutile, anche se in un modo leggermente diverso. Il secondo gettone viene accettato ma non ha effetto sullo stato della macchina: può passare una sola persona e poi il cancelletto torna a bloccarsi. La macchina può accettare anche tre o quattro gettoni di seguito, ma uno solo ha effetto. Forse qualche scettico vorrebbe altre prove prima di accettare la generalizzazione secondo cui tutti i gettoni dopo il primo non hanno effetto; in tal caso, però, dovrà vedersela con i suoi gettoni.

La ragione per cui il cancelletto non può dare più passaggi per più gettoni è che non ha modo di contare i gettoni che riceve. La sua sola forma di memoria è del tutto rudimentale: passando da uno stato all'altro «ricorda» se l'input più recente era un gettone o una spinta sui bracci. Tutti gli input precedenti vanno perduti. Val la pena di notare che questa smemoratezza non va mai a svantaggio della città. Le cose potrebbero andare ben peggio: si potrebbe progettare un cancelletto che cambi di stato dopo ogni moneta, indipendentemente dallo stato attuale, nel quale caso due gettoni in fila non farebbero passare nessuno.

Il cancelletto illustra la maggior parte delle proprietà essenziali di una macchina finita. Ovviamente la macchina deve avere degli stati, che possono essere solo in numero finito. Ci possono essere input e output associati a ogni stato. Gli stati devono essere discreti, cioè chiaramente distinguibili, e le transizioni da uno stato all'altro devono essere efficacemente istantanee. Molto dipende dal punto di vista: giorno e notte sono stati discreti se si vogliono definire l'alba e il tramonto come processi istantanei. La macchina è fatta solo dell'insieme degli stati, degli input e degli output; non ci possono essere dispositivi ausiliari, e in particolare non può esserci alcuna possibilità d'immagazzinamento di informazioni.

Le regole per la costruzione di un automa finito consentono delle varianti. Ci sono automi deterministici e non deterministici, automi di Moore e automi di Mealy. In un automa deterministico, un dato input in un dato stato produce invariabilmente lo stesso risultato; in un automa non deterministico ci possono essere più transizioni possibili. Nell'automa di Moore (dal nome di Edward F. Moore) ogni stato ha un unico output; nell'automa di Mealy (dal nome di G. H. Mealy) gli output sono associati alle transizioni invece che agli stati. Risulta, però, che la varietà di architetture è un po' un'*illusione*: qualsiasi compito possa essere eseguito da un certo tipo di macchina finita può essere eseguito anche dagli altri tipi, anche se può variare il numero di stati necessari. In questa sede parlerò soprattutto degli automi di Moore deterministici, quelli con la struttura più semplice.

Se vi mettete alla ricerca di macchine finite, ne troverete dappertutto. I congegni a moneta sono gli esempi favoriti dei manuali. Alcune macchine emettitrici sono meno rapaci del cancelletto della metropolitana: una volta ricevuta la somma giusta, entrano in uno stato in cui tutte le monete ulteriori sono respinte. Il congegno a moneta con il maggior numero di stati possibile è sicuramente la *slot machine* di Las Vegas. In linea di principio è deterministica, nondimeno è decisamente difficile trovare un input (una moneta e una pressione sulla leva) che provochi una transizione a un particolare stato finale.

Molti apparecchi domestici possono essere visti come automi finiti, sia pure particolarmente ottusi. Una lavabiancheria passa attraverso un'inflessibile successione di stati - riempimento, lavaggio, risciacquo, centrifuga - e i pochi input dotati di significato, come il togliere la spina dalla presa elettrica, hanno di solito lo stesso effetto in tutti gli stati. In modo analogo, un semaforo ha un piccolo repertorio di stati che si ripetono indefinitamente. Il più assillante di tutti gli automi finiti è a mio giudizio un orologio digitale. Se visualizza il mese, il giorno e il passaggio delle ore, dei minuti e dei secondi, ha qualcosa come 31 milioni di stati e nel corso di un anno visita ogni stato esattamente una volta.

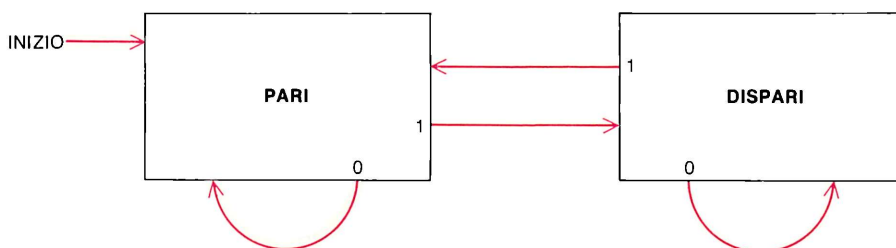
Una trappola per topi è un automa finito: il topo, di solito a suo scapito, innesca una transizione dallo stato di carica allo stato di scatto. Una serratura a combinazione è un automa finito con molti input possibili, uno solo dei quali provoca una transizione di stato. Un telefono ha stati che potrebbero essere definiti agganciato, sganciato, attesa, segnale, composizione del numero, squillo, collegato e fuori uso. Un'automobile può dimostrare efficacemente come l'effetto di un input vari a seconda dello stato del sistema. Che cosa succede quando si preme a fondo l'acceleratore? Dipende. La frizione è inserita? Il freno a mano è sbloccato? La marcia è ingranata? È una marcia avanti o la retromarcia? La porta del garage è aperta?

Nella cellula vivente, il sistema molecolare costituito dal ribosoma e dalle varie specie di RNA di trasporto opera come un automa finito. Gli input sono le quattro basi nucleotidiche dell'RNA

messaggero, indicate con le abbreviazioni *U, A, G* e *C*. Gli output sono i 20 amminoacidi che compongono le proteine. Una catena di nucleotidi è riconosciuta come input valido per l'automa solo se inizia con il segnale di «partenza» *AUG*. In seguito l'automa legge in modo continuo il flusso di input, cambiando stato ogni volta che riconosce un codone, ossia una tripletta di nucleotidi. I tre codoni speciali *UAA, UAG* e *UGA* sono segnali di «fine»: quando ne incontra uno, l'automa si ferma. Molti altri sistemi biologici possono essere rappresentati come automi finiti; esempi che vengono in mente sono la molecola di emoglobina e le proteine promotore e repressore dei batteri.

Nella teologia di Tommaso d'Aquino l'anima è un automa finito, meravigliosamente elaborato e totalmente deterministico. È creata in uno stato di rischio, come conseguenza del peccato originale. Con il battesimo entra in uno stato di grazia, ma certi atti (idolatria, bestemmia, adulterio e così via) inducono una transizione a uno stato di peccato. Sono allora necessari confessione, pentimento e assoluzione per riportare l'anima allo stato di grazia. L'effetto di un input finale, la morte, dipende tutto dallo stato dell'anima al momento della morte: in uno stato di grazia la morte porta alla salvezza ma in uno stato di peccato porta alla dannazione. La macchina anima è in realtà più complessa di quanto faccia pensare questa descrizione: bisognerebbe distinguere tra i vari gradi di peccato (veniale e capitale, attuale e abituale) e si dovrebbe tener conto di altri possibili stati dell'anima (come quelli associati al limbo e al purgatorio) e di altri possibili input (come il Giudizio Universale).

Nella meccanica quantistica anche l'atomo diviene un automa finito, quindi lo stesso avviene per ogni cosa fatta di atomi. Gli stati dell'atomo sono i livelli di energia consentiti; gli input e gli output sono i fotoni, quanti di radiazione elettromagnetica. In una descrizione accurata, penso che l'atomo sarebbe classificato come automa di Mealy non deterministico con transizioni epsilon. È una macchina non deterministica perché l'effetto di un input non può essere previsto con certezza. È un automa di Mealy perché la natura dell'output (vale a dire l'energia del fotone) è determinata dalla transizione, non dallo stato in cui la macchina è entrata. Le transizioni epsilon sono quelle che possono aver luogo in assenza di qualsiasi input; devono essere incluse nel modello perché un atomo può emettere un fotone e cambiare di stato del tutto spontaneamente.

Un diagramma delle transizioni di stato per un cancelletto di metropolitana

La macchina per il controllo della parità

Il cervello è un automa finito? Per coincidenza, i moderni studi sui sistemi finiti iniziarono proprio con un modello di reti di neuroni ideato nel 1943 da Warren S. McCulloch e Walter Pitts. I neuroni di McCulloch e Pitts erano semplici cellule con input eccitanti e inibenti; ogni cellula aveva un solo output e due stati interni: eccitato e non eccitato. Le cellule potevano essere disposte in reti per compiere varie funzioni logiche, tra cui le funzioni «and», «or» e «not» che sono ora elementi di uso comune nei sistemi logici elettronici. Stephen C. Kleene, dell'Università del Wisconsin a Madison, dimostrò nel 1956 l'equivalenza tra le reti di neuroni ideali e i diagrammi delle transizioni di stato che abbiamo illustrato qui.

Quarant'anni dopo il lavoro di McCulloch e Pitts, è ancora argomento di discussione la possibilità di classificare il cervello tra i sistemi finiti. Naturalmente, il numero di neuroni è necessariamente finito, ma questo non è il solo problema. Un vero neurone è molto più complesso di una cellula a due stati e alcune delle sue proprietà possono variare con continuità, anziché essere vincolate a occupare stati discreti. Inoltre, il divieto di un magazzino ausiliario di informazioni in un modello a stati finiti è quanto mai imbarazzante. Se la vita mentale non è altro che una successione di stati istantanei, senza conoscenza della propria storia, allora che cos'è la memoria?

Gli stati della mente di cui si occupa la psicologia, quali noia, paura, desiderio, gioia e dolore, sembrano rientrare più facilmente nel complesso di una teoria a stati finiti. D'altra parte, gli stati sono così numerosi e così limitata è la comprensione delle transizioni, che il modello sembra inutilizzabile. Solo per animali di minor complessità è possibile tracciare più di qualche isolato frammento del diagramma delle transizioni di stato e in quelle specie lo sperimentatore può non avere alcun accesso diretto ai presunti stati mentali. In effetti, questa linea di ricerca è stata seguita soprattutto dai comportamentisti, che negano l'effettiva esistenza di stati mentali.

Anche il caso del calcolatore digitale - e intendo qui la macchina tangibile, l'hardware - è problematico. Il comune modello mentale di un calcolatore, formulato da John von Neumann, divide la macchina in una unità centrale di elaborazione e una schiera di celle di memoria. Il concetto di stati finiti è indubbiamente applicabile alle varie componenti dell'unità centrale di elaborazione quali registri, sommatori e il meccanismo di controllo deputato a dirigere le operazioni interne dell'unità di elaborazione.

I guai iniziano quando si prende in considerazione la memoria. Secondo le regole per la costruzione di un automa finito, non è consentita alcuna memoria esterna, quindi ogni cella deve essere vista non come un elemento di immagazzinamento separato dall'unità di elaborazione, ma come una parte dello stato complessivo della macchina. Se tutte le celle sono vuote, il calcolatore è in uno stato; se si riempie una sola cella, entra in un altro stato, e così via. Questa concezione del calcolatore è ben poco illuminante, in parte perché non stabilisce alcun collegamento tra lo stato della macchina e ciò che essa sta facendo. Inoltre, il numero di stati è immenso. Forse nemmeno un calcolatore di modeste dimensioni (100 elementi binari), che avesse girato continuamente per tutta l'età dell'universo, sarebbe passato per tutti i propri stati.

Il ruolo fondamentale dell'automa finito nella scienza del calcolatore è a un livello d'astrazione maggiore dei meccanismi da orologeria dell'hardware. Un calcolatore che giri sotto la direzione di un programma non è più un insieme di porte logiche, registri, celle di memoria e altri aggeggi elettronici; è una macchina «virtuale» le cui parti funzionanti sono definite dal programma e possono essere ridefinite se necessario. Mentre l'hardware conosce solo gli interi binari e semplici istruzioni per trasferirli e manipolarli, il calcolatore virtuale ha a che fare con sistemi simbolici molto più espressivi: parole, equazioni, matrici, funzioni, vettori, codoni, liste, immagini, magari addirittu-

Gli stati dell'anima nella teologia di Tommaso d'Aquino

ra idee. Le tecniche a stati finiti possono essere di qualche validità per la creazione del calcolatore virtuale e a volte il calcolatore virtuale *è* un automa finito.

Consideriamo un programma il cui obiettivo sia leggere una serie di cifre binarie (1 o 0) e riferire se il numero di 1 ricevuti sia pari o dispari. (Questo compito ha un significato pratico; programmi di controllo della parità sono impiegati, per esempio, nell'individuazione di errori nella trasmissione telefonica di dati digitali.) Il programma può essere costruito come un automa finito con due stati, come si vede nella figura in basso della pagina precedente. L'operazione inizia nello stato pari perché nessun 1 è stato ricevuto e 0 è considerato un numero pari. Ogni 1 nel flusso di input provoca un cambiamento di stato, mentre uno 0 ricevuto nell'uno o nell'altro stato lascia immodificato lo stato stesso. Anche se la macchina non può «ricordare» alcun input che preceda quello più recente e certamente non può contare gli 1 o gli 0, il suo output riflette sempre la parità del flusso di input.

Il modello a stati finiti del calcolo si ritrova comunemente nei programmi che hanno in qualche modo a che fare con testi o altre informazioni sotto forma linguistica. L'esempio più rilevante si trova nei compilatori: programmi che traducono enunciati di programmazione formulati in un linguaggio sorgente in enunciati equivalenti formulati in un linguaggio oggetto, per lo più il «linguaggio macchina» di un particolare calcolatore. I compilatori e altri programmi di traduzione sono essenziali alla nozione di macchina virtuale, in quanto mediano tra simboli con un significato per l'uomo e quelli riconosciuti dal calcolatore.

La parte di compilazione che si può designare come automa finito è detta analizzatore lessicale o *scanner.* Come il cancelletto della metropolitana, è una macchina inghiotti-gettoni. In questo caso, però, i gettoni sono le parole, le unità lessicali fondamentali, del linguaggio. L'analizzatore esamina ogni gruppo di caratteri e stabilisce se si tratta di un vero «gettone», cioè una parola, come una istruzione o un numero; se non lo è, l'analizzatore lo respinge come privo di senso, proprio come il cancelletto respingerebbe un gettone falso.

L'attività di un analizzatore lessicale può essere illustrata da un automa finito impiegato per riconoscere le parole di un semplice linguaggio, anche se di dominio espressivo limitato: le parole sono fatte esclusivamente di numerali romani. Sono accettati, in realtà, solo numerali romani di forma particolare: devono essere in stretta notazione additiva, così che il 9 è rappresentato da VIIII invece che da IX. (Sembra che gli stessi romani impiegassero la notazione additiva e si ritiene che la forma sottrattiva sia stata un'innovazione germanica.)

Nella figura della pagina a fronte si vede un diagramma delle transizioni di stato per la macchina a numerali romani. Il suo alfabeto di simboli di input comprende le lettere M, D, C, L, X, V, I e inoltre il simbolo di spazio. Tutti gli spazi iniziali sono semplicemente ignorati, ma una volta ricevuta la prima lettera il programma compie un'immediata transizione a uno stato identificato (per convenienza) dal nome della lettera. Se la prima lettera è una M, può essere seguita da qualsiasi carattere appartenente all'insieme accettato, inclusa un'altra M. Se il carattere successivo è una D, però, la situazione è diversa. Dallo stato D non è definita alcuna transizione che riporti allo stato M, perché qualsiasi serie di simboli che comprenda DM non può essere una parola ben formata nel linguaggio dei numerali additivi romani. Inoltre, non c'è transizione dallo stato D allo stato D stesso, quindi anche DD è una sequenza esclusa. (La ragione è che i simboli di «mezzo valore» D, L e V non possono essere ripetuti nei numerali romani corretti.)

Nello stato D, le uniche lettere accettate sono quelle di valore inferiore: C, L, X, V e I. Lo stesso insieme è accettato nello stato C (perché C può essere ripetuto), ma nello stato L solo le lettere X, V e I sono riconosciute. Dovrebbe essere chiara la regola che governa le transizioni. Gli stati sono disposti in una gerarchia e una

volta raggiunto un certo livello la macchina non può mai tornare a un livello superiore; nei livelli di mezzo valore non può mai rimanere allo stesso livello. Quando è raggiunto lo stato I è consentito solo un altro I o uno spazio. Lo spazio, immesso a questo punto o in qualsiasi altro momento dopo la prima lettera, indica la fine della parola e rimanda la macchina allo stato di partenza, pronta a ricevere il successivo numerale romano.

Nessun linguaggio di programmazione a me noto consente l'immissione di numeri in forma romana, ma virtualmente tutti questi linguaggi hanno la possibilità di maneggiare le cifre arabe. Le tecniche per il riconoscimento sono analoghe, anche se c'è una maggiore varietà di formati. Interi semplici come 137 possono essere maneggiati, in linea di principio, da una macchina a uno stato, ma le molteplici parti di un numero come $+6{,}625 \times 10^{-27}$ richiedono un'analisi lessicale più elaborata.

Il sistema ribosoma-RNA di trasporto può essere visto come un analizzatore lessicale che riconosce le sequenze di nucleotidi biologicamente significative in una molecola di RNA messaggero. Per essere accettata, una sequenza deve iniziare con un codone di inizio e terminare con uno dei tre codoni di fine; all'interno di questi confini, è consentita qualsiasi combinazione dei simboli di input *U, A, G* e *C,* presi a tre a tre.

L'analisi lessicale è solo il primo passo nel processo della compilazione. I componenti del compilatore messi in azione dopo il dispositivo di analisi lessicale sono l'analizzatore sintattico o *parser* e il generatore di codice. L'analizzatore sintattico assume come input i gettoni identificati dall'analizzatore lessicale e analizza le loro relazioni sintattiche; è qui che il compilatore si avvicina maggiormente a capire il significato degli enunciati del programma che traduce. Il generatore di codice scrive un programma nel linguaggio oggetto che esegue le funzioni definite dagli enunciati analizzati.

Per i linguaggi giocattolo considerati qui, i compiti dell'analizzatore sintattico e del generatore di codice sono banali. La forma compilata di un enunciato nel linguaggio dei numerali romani potrebbe essere semplicemente l'equivalente arabo del numero e potrebbe essere generata dalla seguente strategia. Prima che una parola sia sottoposta ad analisi lessicale, si specifica una cella di immagazzinamento, che viene posta uguale a zero. Poi, ogni volta che l'analizzatore lessicale entra nello stato M si aggiunge 1000 al valore della cella; per lo stato D si aggiunge 500 e così via. Una volta completata l'analisi lessicale, la cella di memoria contiene il valore del numerale romano. Si noti che il compilatore giocattolo non è più un puro automa finito perché possiede un dispositivo di immagazzinamento ausiliario.

Un compilatore per il codice genetico è ancora più semplice e può essere interamente realizzato nel contesto di un sistema a stati finiti. Il programma «compila-

Un analizzatore lessicale per un linguaggio di numerali romani

Un automa finito traduce il codice genetico in proteina

to» è una sequenza dei simboli di tre lettere standard per gli amminoacidi; i simboli possono essere generati come output degli stati dell'analizzatore lessicale che riconosce i codoni. I tre stati corrispondenti ai codoni di stop non hanno output.

La creazione di un compilatore per un linguaggio abbastanza esteso da essere di utilità generale non è frutto del caso: l'architettura sottostante all'automa finito fornisce almeno un principio di organizzazione. Se la sintassi del linguaggio è specificata con sufficiente precisione, parte del lavoro può anche essere meccanizzata: può essere svolta da un compilatore di compilatore, un programma che ha per input una descrizione formale di un linguaggio e per output un altro programma che traduce enunciati nel linguaggio. Per quanto ne so, nessuno ha ancora pensato a scrivere un compilatore di compilatore di compilatore.

L'identificazione di parole da parte di un analizzatore lessicale è in se stessa un tipo di analisi e l'insieme di tutte le possibili sequenze di simboli in una parola è un tipo di linguaggio. È in realtà un linguaggio infinito: a meno di porre qualche limite artificiale alla lunghezza delle sequenze individuali, si può formare un'infinita varietà di parole riconoscibili. Come fa una macchina con un numero finito di parti a riconoscere un'infinità di enunciati ben formati e a escluderne un'infinità di mal formati? La chiave sta nella struttura del linguaggio stesso. Se gli enunciati di un linguaggio infinito vanno riconosciuti da un automa finito, devono essere formati secondo regole rigide.

Le regole furono enunciate da Kleene nel 1956; esse definiscono una classe di linguaggi detti linguaggi regolari o insiemi regolari. Kleene dimostrò che un automa finito può riconoscere un linguaggio solo se è regolare e, inoltre, che ogni linguaggio regolare può essere riconosciuto da qualche automa finito. Cosa si intenda per regolare può essere indicato brevemente (anche se in modo non rigoroso) da due regole. Primo, qualsiasi linguaggio finito è regolare e può quindi essere riconosciuto da un automa finito; dopo tutto, si potrebbe costruire una macchina con uno stato per ogni possibile espressione del linguaggio. Secondo, se un linguaggio è infinito deve essere possibile analizzare sintatticamente tutti i suoi enunciati leggendo un simbolo alla volta da sinistra a destra, ossia dall'inizio alla fine, senza mai tornare indietro o guardare in avanti. Se l'accettabilità di un simbolo dipende dalla presenza di un altro simbolo, deve trattarsi del simbolo immediatamente a sinistra.

La seconda regola è una diretta conseguenza delle limitazioni di un automa finito, che non può né prevedere i suoi stati futuri né conservare un ricordo di quelli passati; deve scegliere una transizione di stato che si basi solo sullo stato attuale e sull'attuale simbolo di input. È per questa ragione che un automa finito non può maneggiare una notazione sottrattiva per i numerali romani. Se l'espressione XI

La gerarchia di Chomsky di macchine finite e infinite

viene letta e la macchina la interpreta come 11, non può tornare a rivedere il valore quando il carattere successivo risulta essere V. Molte altre funzioni sono governate dalla stessa limitazione. Per esempio, non è possibile costruire un automa finito che legga una sequenza di cifre binarie e stabilisca se il numero di 1 è uguale al numero di 0. Analogamente, sebbene un automa finito possa sommare numeri binari, non può moltiplicarli; lascio al lettore capire perché.

Al di là degli automi finiti e dei linguaggi regolari si estende una gerarchia di macchine più potenti e di linguaggi più generali. Tale gerarchia è detta gerarchia di Chomsky, dal nome del linguista Noam Chomsky che studiò i vari linguaggi formali come possibili modelli per il linguaggio naturale. Linguaggi più generali si creano allentando le limitazioni sulle regole grammaticali degli insiemi regolari; le macchine sono costruite aggiungendo elementi di memoria al modello base a stati finiti.

La prima macchina della serie è detta macchina finita con memoria di tipo *pushdown*. Consiste di un automa finito con l'aggiunta di una matrice di memoria con una capacità infinita ma una organizzazione particolare. La memoria prende la forma di una catasta, come una catasta di vassoi in un self-service. Un elemento di informazione può essere immagazzinato solo ponendolo in cima alla catasta e per recuperarlo bisogna prima rimuovere tutti gli elementi che gli stanno sopra. In questo modo, l'ultimo elemento entrato è il primo a uscire.

Il linguaggio riconosciuto da una macchina finita con memoria di tipo *pushdown* è detto linguaggio libero da contesto. Analizzando sintatticamente i suoi enunciati, l'accettabilità di un simbolo può dipendere sia dal simbolo immediatamente a sinistra sia da quello immediatamente a destra. Questa dipendenza bidirezionale è ammissibile perché qualsiasi simbolo la cui interpretazione non possa essere decisa immediatamente può essere immagazzinato nella catasta finché l'ambiguità è risolta. Una macchina finita con memoria di tipo *pushdown*, quindi, può lavorare con numeri romani sottrattivi e può identificare espressioni con numeri uguali di 1 e 0 (o altri simboli, quali parentesi aperta e chiusa). Invece, non può individuare enunciati con numeri uguali di tre simboli (per esempio 0, 1, 2). La maggior parte dei linguaggi di programmazione sono liberi da contesto e l'analizzatore sintattico di un compilatore è di solito una macchina finita con memoria di tipo *pushdown*. Molti calcolatori hanno dispositivi hardware per organizzare parte della capacità di memoria come catasta di tipo *pushdown*. Un linguaggio di programmazione, il Forth, fa di una catasta la struttura primaria di memoria. Naturalmente, in una macchina reale una catasta non può avere lunghezza infinita.

I linguaggi liberi da contesto hanno un nome adeguato in quanto l'analisi sintattica di un simbolo può essere influenzata direttamente solo dai due simboli immediatamente adiacenti, e non dal più ampio contesto in cui esso si trova. Togliendo questa limitazione si ottiene un linguaggio sensibile al contesto e aumenta ulteriormente la difficoltà di interpretazione. Ora possono interagire simboli molto distanti uno dall'altro; nel caso peggiore non è possibile interpretare il primo simbolo di un'espressione finché non è stato letto l'ultimo. A compenso della maggiore complessità, si guadagna qualcosa in capacità d'azione. Una macchina basata su un linguaggio sensibile al contesto può stabilire se in un'espressione si trovano numeri uguali di tre simboli.

La macchina che può riconoscere un linguaggio sensibile al contesto è un automa limitato linearmente. Oltre al consueto apparato di automa finito, ha una memoria organizzata in modo che in qualsiasi momento si possa raggiungere qualsiasi locazione di immagazzinamento; è una macchina ad accesso casuale. La memoria ha una capacità finita, ma si presuppone che sia abbastanza grande da contenere qualsiasi input la macchina riceva. L'automa limitato linearmente sembra una buona approssimazione al modello di von Neumann di calcolatore digitale. Stranamente, però, i corrispondenti linguaggi di programmazione sensibili al contesto sembrano rari; evidentemente, la più semplice struttura libera da contesto ha quasi sempre sufficiente potenza espressiva.

Tutti i linguaggi descritti sopra hanno una proprietà in comune: sono detti ricorsivi. Con questa designazione si intende che si può immaginare una procedura per generare tutte le possibili «espressioni» del linguaggio in ordine di lunghezza crescente. Ne consegue che esiste un metodo per decidere se un dato enunciato di lunghezza finita è un membro del linguaggio: basta generare tutti gli enunciati fino a quella lunghezza e confrontarli.

Vi sono linguaggi che non possono soddisfare nemmeno questo standard minimo di trattabilità. C'è solo una macchina che possa riconoscerli: è l'ultima spiaggia del calcolatore, la macchina di Turing, un automa finito che può spaziare liberamente in una memoria senza limiti. Nella descrizione data da Turing, la memoria è un nastro, infinito in entrambe le direzioni e diviso in celle su cui l'apparato a stati finiti può scrivere, leggere o cancellare.

Guardando dall'elevato punto di vista della macchina di Turing, si chiariscono le relazioni tra i più modesti congegni di calcolo. L'automa limitato linearmente è semplicemente una macchina di Turing con un nastro finito. La macchina finita con memoria di tipo *pushdown* ha un nastro infinito in una direzione, ma la «testina» per leggere e scrivere sul nastro rimane sempre fissa sull'ultima cella non vuota. L'automa finito è una macchina di Turing del tutto priva di nastro.

Forse i lettori sempre a caccia di novità, ansiosi di analizzare sintatticamente linguaggi non ricorsivi, sono già usciti per acquistare una macchina di Turing. Andrebbero avvertiti che anche il calcolatore «estremo» ha le sue debolezze. Ci sono linguaggi con grammatiche così strampalate che nemmeno con una macchina di Turing si potrebbero riconoscere i loro enunciati in un tempo finito. Finora questi linguaggi hanno trovato scarso uso nel mondo delle macchine da calcolo, ma la gente riesce in qualche modo a parlarli.

L'automa cellulare offre un modello del mondo e un mondo in se stesso

di Brian Hayes

Le Scienze, maggio 1984

È davvero sorprendente che le molecole d'acqua «sappiano» come formare le elaborate simmetrie di un fiocco di neve. Non c'è architetto a dirigerne le combinazioni e le molecole non hanno uno «stampo» per la forma cristallina. La configurazione su larga scala emerge completamente dalle interazioni a breve raggio di molte unità identiche. Ogni molecola risponde solo all'influenza dei suoi immediati adiacenti, tuttavia una disposizione coerente si mantiene in una struttura fatta forse di 10^{20} molecole.

Un modo per avvicinarsi alla comprensione di questo processo consiste nell'immaginare che ogni sito in cui può prender posto una molecola sia governato da un rudimentale calcolatore. Col crescere del cristallo, ogni calcolatore passa in rassegna i siti circostanti e, a seconda di ciò che trova, stabilisce attraverso qualche regola prefissata se il proprio sito debba essere occupato o rimanere vuoto. Lo stesso calcolo viene fatto in tutti i siti secondo la stessa regola.

Il modello computazionale della crescita di un fiocco di neve è un automa cellulare: una schiera uniforme di numerose celle identiche, in cui ogni cella può assumere solo pochi stati e interagisce solo con poche celle adiacenti. I componenti del sistema (le celle e la regola per calcolare lo stato successivo di una cella) possono essere davvero semplici e nondimeno dar luogo a un'evoluzione notevolmente complessa.

L'idea dell'automa cellulare è vecchia approssimativamente quanto il calcolatore elettronico digitale. Le prime ricerche furono condotte da John von Neumann (con un importante contributo di Stanislaw Ulam) nei primi anni cinquanta. L'obiettivo principale di von Neumann era escogitare un semplice sistema capace di riprodursi alla maniera di un organismo vivente. Anche il più noto automa cellulare, il gioco «Vita» inventato nel 1970 da John Horton Conway, ha un aspetto biologico, come suggerisce il nome; le cellule nascono, vivono o muoiono a seconda della densità della popolazione locale.

Nelle ricerche più recenti sugli automi cellulari, il centro di interesse si è leggermente spostato. Le schiere di celle a interazione locale sono viste come modelli potenzialmente utili di sistemi fisici, dai fiocchi di neve ai ferromagneti alle galassie. Inoltre possono trovare applicazioni in problemi della scienza dei calcolatori, sia pratica (come si dovrebbe organizzare una rete di molti calcolatori interagenti?) sia teorica (qual è il limite ultimo alla potenza di una macchina per il calcolo?). La cosa forse più interessante è che l'automa cellulare può essere visto come un «universo digitale» che val la pena di esplorare per se stesso, al di là della sua utilità come modello del mondo reale.

Il risorgere dell'interesse per gli automi cellulari è stato evidenziato da un seminario tenuto sull'argomento nel 1983 al Los Alamos National Laboratory. Gli atti (circa 20 articoli) sono stati in seguito pubblicati in «Physica D» e, in forma di libro, dalla North-Holland Publishing Company. Il contenuto del presente articolo si basa quasi tutto su quanto discusso nell'incontro di Los Alamos.

Quattro proprietà caratterizzano un automa cellulare. La prima è la geometria della matrice di celle. Per un modello della crescita di un fiocco di neve sarebbe adeguata una matrice esagonale a due dimensioni, ma nella maggior parte dei contesti si sceglie un reticolo rettangolare fatto di quadrati identici. È facile costruire matrici in tre o più dimensioni, ma non è facile visualizzarle. Recentemente sono state fatte sorprendenti scoperte con l'ancor più semplice matrice unidimensionale: una semplice fila di celle.

All'interno di una matrice data è necessario specificare l'intorno che ogni cella esamina nel calcolo del suo stato successivo. Nella matrice rettangolare bidimensionale si è prestata molta attenzione a due intorni. Von Neumann limitava l'attenzione di ogni cella alle quattro adiacenti più vicine, quelle a nord, sud, est e ovest; questo insieme di celle è chiamato ora intorno di von Neumann. L'intorno che comprende queste quattro celle e le quattro diagonalmente adiacenti è detto intorno di Moore, dal nome di Edward F. Moore. Ovviamente gli intorni si sovrappongono e una data cella è inclusa simultaneamente negli intorni di parecchie celle adiacenti. In qualche caso la cella di centro - la cella che effettua un calcolo - è considerata membro del suo stesso intorno.

Il terzo fattore da prendere in considerazione per descrivere un automa cellulare è il numero di stati per cella. Von Neumann aveva scoperto una configurazione autoreplicantesi fatta di celle con 29 stati possibili, ma la maggior parte degli automi sono molto più semplici. C'è, in effetti, un'ampia possibilità di variazione anche tra gli automi binari, quelli con due stati per cella; gli stati possono essere rappresentati come 1 o 0, vero o falso, acceso o spento, vivo o morto.

La principale fonte di varietà nell'universo degli automi cellulari è l'enorme numero delle regole possibili, per stabilire il futuro stato di una cella, basate sull'attuale configurazione del suo intorno. Se k è il numero di stati per cella e n è il numero di celle incluse nell'intorno, vi sono k^{k^n} possibili regole. Per un automa binario, quindi, nell'intorno di von Neumann (dove n è 4) ci sono più di 65 000 possibili regole; nell'intorno di Moore (dove n è 8) ve ne sono 10^{77}. Solo una frazione insignificante di tali regole è stata esaminata.

Il gioco Vita si effettua con celle a due stati su un reticolo rettangolare nell'intorno di Moore, con la complicazione che la cella centrale è significativa. In altri termini, a ogni passo dell'evoluzione del sistema ogni cella controlla sia lo stato delle otto celle circostanti sia il proprio. Secondo la regola definita da Conway, se la cella centrale è viva continuerà a vivere nella generazione successiva se due o tre delle otto celle dell'intorno sono anch'esse vive. Se ci sono tre celle vive nell'intorno, la cella centrale è viva nella generazione successiva indipendentemente dal suo stato attuale. In tutti gli altri casi, la cella centrale muore o rimane morta.

Il fascino del gioco Vita sta nella sua imprevedibilità. Alcune configurazioni muoiono completamente; molte di più cadono in una configurazione stabile o in una ciclica con un periodo di poche generazioni. Nel corso degli anni, comunque, si sono scoperti un certo numero di stati iniziali più interessanti, come il «cannone ad alianti», che lancia un fiume interminabile di proiettili. L'esplorazione dei meandri della vita continua. Alcuni recenti sviluppi sono descritti da Martin Gardner in *Wheels, Life, and Other Mathematical Amusements* (pubblicato da Freeman & Co., New York, nel 1983). Vorrei ora passare ad altri automi cellulari con proprietà che solo da poco cominciano a essere chiarite.

Tra la moltitudine di possibili regole di transizione, molte sono di scarso interesse intrinseco. Per esempio, una regola che stabilisce che una cella sarà «accesa» se e solo se la cella alla sua sinistra è accesa determina un'evoluzione decisamente facile da prevedere: una qualsiasi configurazione iniziale conserva la sua forma ma si sposta a destra di una cella a ogni passo nel tempo. Una sottoclasse di regole dette regole di conto o regole totalistiche sembra includere esempi di quasi tutte le varietà osservate di automi cellulari. Con regole di questo tipo, il nuovo stato di una cella dipende solo dal numero di

adiacenti in un dato stato, non dalla loro posizione. Molti automi basati su queste regole sono stati studiati da membri dell'Information Mechanics Group del Laboratory for Computer Science al Massachusetts Institute of Technology. Il gruppo è formato da Edward Fredkin, Norman Margolus, Tommaso Toffoli e Gérard Y. Vichniac.

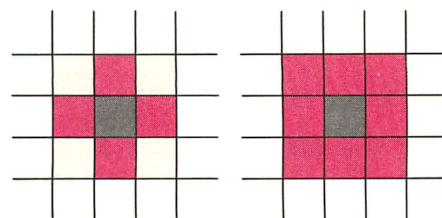
Una delle più semplici regole di conto è la regola della parità, che assegna il valore 1 a una cella se un numero dispari di celle adiacenti hanno valore 1; in caso contrario le assegna il valore 0. L'evoluzione di questo sistema, quando la regola è applicata nell'intorno di von Neumann, è stata descritta in questo volume nell'articolo a pagina 66. Qualsiasi configurazione di partenza è replicata quattro volte; le quattro copie sono poi replicate a loro volta, e così via.

Un'altra classe di regole di conto è costituita dalle regole «di votazione», che danno il valore 1 alla cella centrale ogniqualvolta il numero degli 1 nell'intorno supera una certa soglia. Vichniac, in un saggio presentato all'incontro di Los Alamos, sottolinea come regole di questo tipo diano modelli della percolazione e della nucleazione, fenomeni importanti nella fisica dello stato solido e in altri campi. Percolazione è il termine adottato per la formazione di un percorso ininterrotto attraverso uno spazio; per esempio, quando un metallo si disperde in una matrice isolante, la conducibilità del materiale composito dipende dalla probabilità di formare una catena continua di atomi di metallo, come la trasmissione di una malattia infettiva è possibile attraverso una sequenza ininterrotta di individui adatti. La nucleazione è il processo che dà inizio alla crescita di un cristallo, all'ebollizione di un liquido e a eventi analoghi.

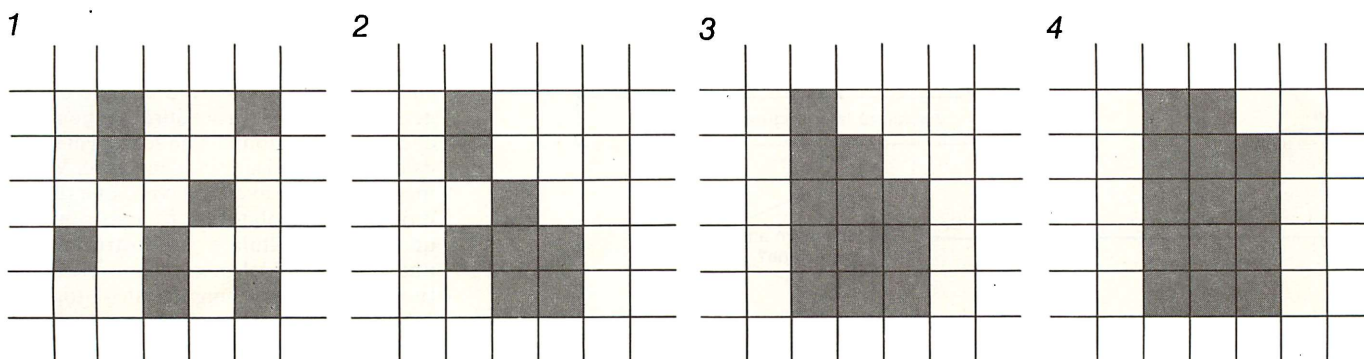
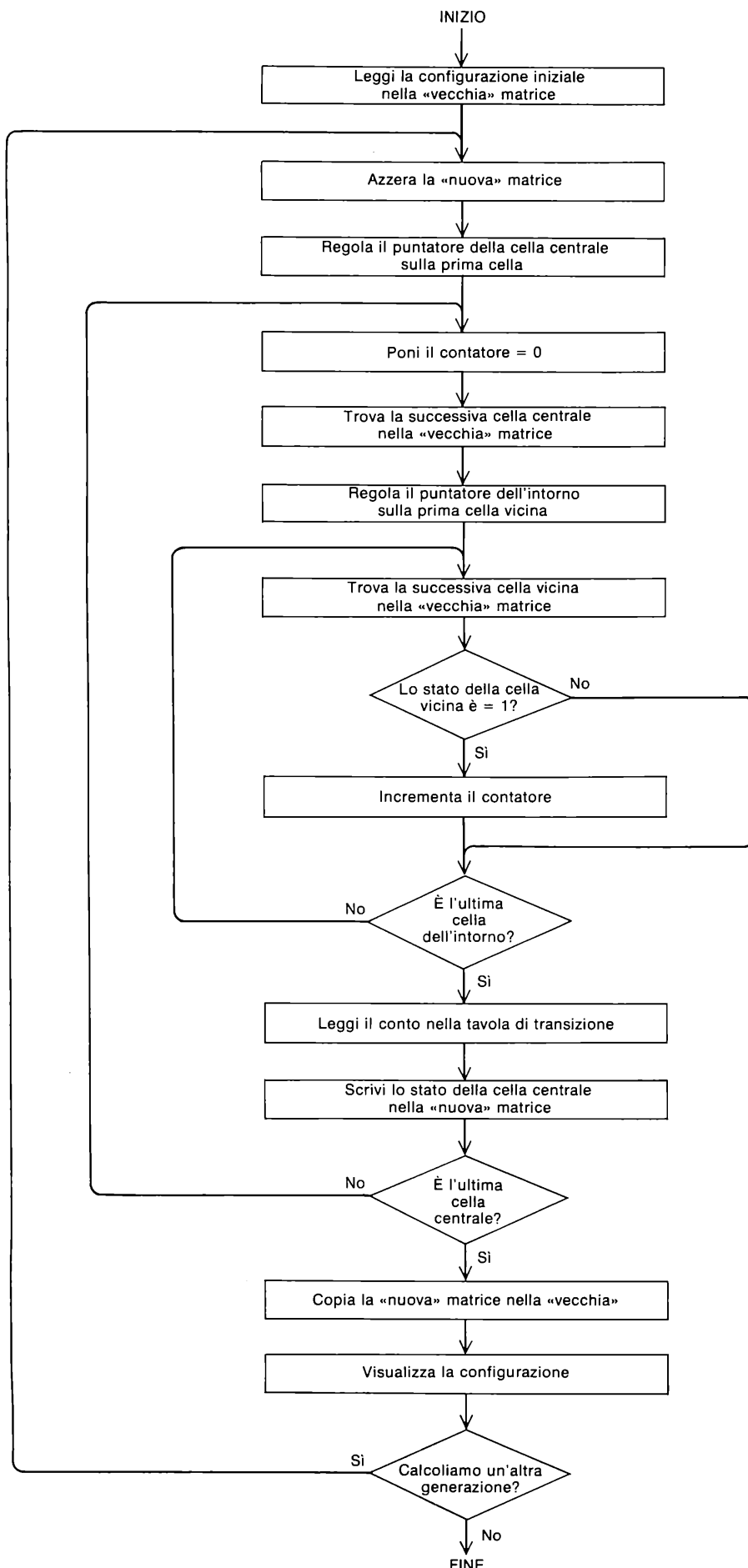
Una regola di transizione che dà origine alla percolazione assegna il valore 1 alla cella centrale se vi sono degli 1 in almeno tre delle cinque celle dell'intorno di Von Neumann più la cella centrale. L'avvio della percolazione è estremamente sensibile alla concentrazione iniziale di 1. Se la concentrazione è inferiore a un mezzo, probabilmente non si formeranno, nel corso dell'evoluzione, catene continue di 1 attraverso la matrice. Con una concentrazione pari o superiore a un mezzo, le catene appaiono ma il reticolo non si riempie ancora totalmente di 1 e rimangono isole di 0 nello stato stabile finale. Si incontra la nucleazione, con la matrice che si riempie completamente di 1, quando la regola è modificata in modo da richiedere solo due 1 su cinque celle. La concentrazione critica è pari a 0,0822.

Il modello di Ising è uno strumento concettuale della fisica che appare superficialmente molto simile a un automa cellulare. Il modello è un reticolo rettangolare in cui ogni sito ha due valori possibili e interagisce solo con i suoi quattro adiacenti più vicini. Si impiega spesso questo modello per descrivere materiali ferromagnetici; ogni sito rappresenta uno spin atomico che deve puntare o verso l'alto o verso il basso. Al di sotto di una temperatura critica (la temperatura di Curie) gli spin tendono a essere allineati e il materiale è allora magnetizzato, ma a temperature superiori sono distribuiti più o meno casualmente.

Nell'articolo a pagina 66 ho parlato di una versione del modello di Ising creata con un programma di tabellone elettronico; il suo reticolo di celle si presta in modo naturale a essere studiato in termini di automi cellulari, per quanto si tratti di un reticolo con regole probabilistiche per simulare la temperatura. Avevo osservato un fenomeno curioso: a bassa temperatura gli spin non assumevano un allineamento uniforme in una direzione e adottavano invece una configurazione a scacchiera di spin alternativamente rivolti verso l'alto e verso il basso. A ogni passo nel tempo tutti gli spin si ribaltavano. In un ferromagnete, la disposizione a scacchiera è la configurazione di massima energia e dovrebbe quindi essere instabile; è la disposizione caratteristica di un antiferromagnete.

Vichniac aveva già individuato il problema e l'aveva spiegato. Nella realizzazione standard del modello di Ising, a ogni iterazione solo uno spin può cambiare. Ne consegue che quando un particolare sito passa in rassegna il suo intorno, alcuni degli spin esaminati sono «vecchi» e alcuni sono «nuovi». In queste condizioni, non può nascere l'oscillante antiferromagnete. Solo quando vengono ricalcolati simultaneamente tutti gli spin viene favorito l'antiferromagnete ad alta energia. Vi sono strategie per evitare questa «catastrofe da retroazione», ma la lezione più significativa da trarne è che è fuorviante attenersi alla più semplice corrispondenza intuitiva tra il modello di Ising e gli automi cellulari.

Gli intorni di von Neumann e di Moore

Vichniac e altri del gruppo del MIT rilevano che gli automi cellulari hanno uno *status* fondamentalmente differente da quello di altri modelli fisici. Lo stratagemma più comune per costruire un modello matematico del mondo naturale è stato a lungo l'equazione differenziale, che può descrivere il cambiamento in una certa grandezza come funzione della posizione e del tempo. Per esempio, le equazioni di Maxwell forniscono la variazione del valore di un campo elettromagnetico da punto a punto e da istante a istante. Tutte le grandezze in queste equazioni variano con continuità. Un automa cellulare, invece, è un sistema totalmente discreto. Lo spazio non è un continuum ma una matrice di celle; anche il tempo è spezzettato in passi discreti e mentre l'intensità di un campo può variare su un dominio continuo, le celle di un automa cellulare possono avere solo un numero finito di stati.

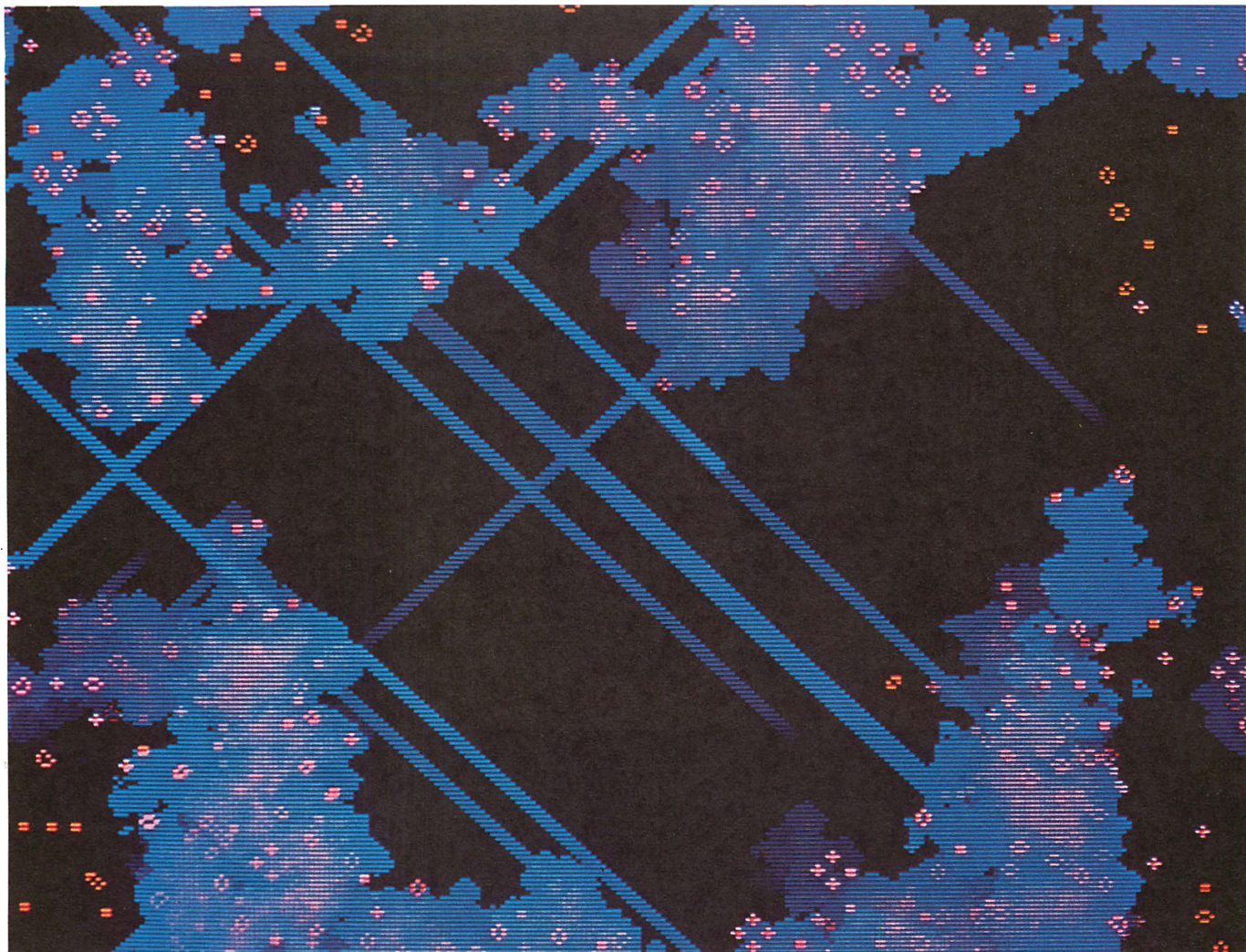
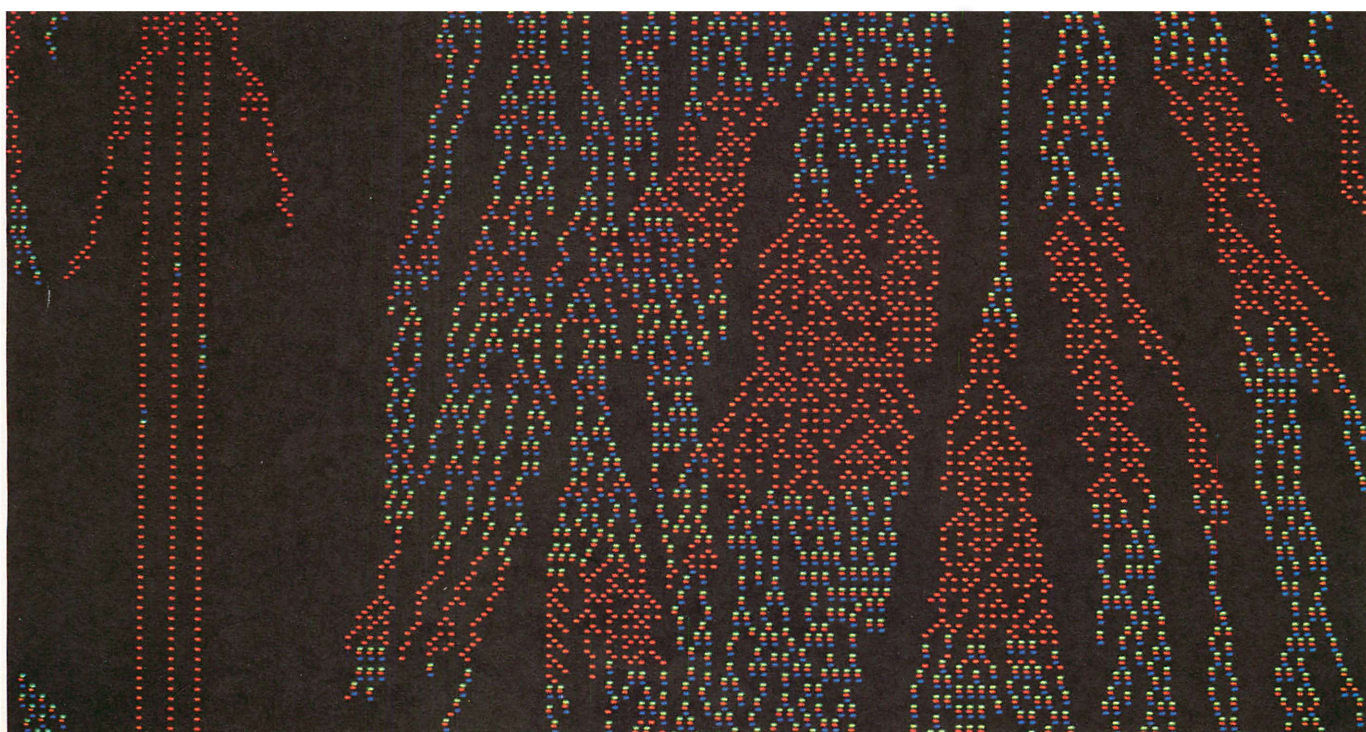
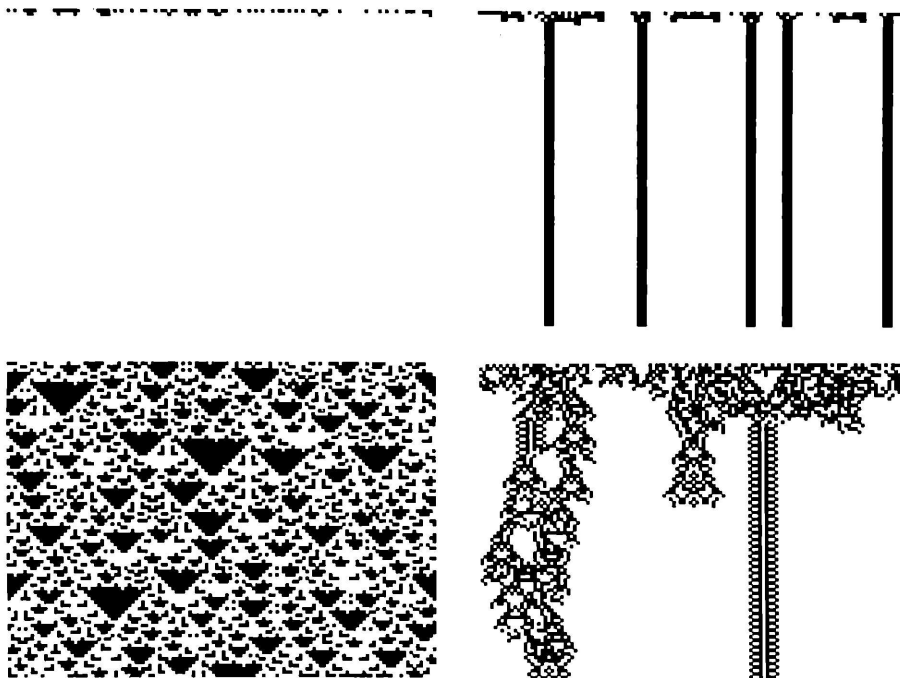
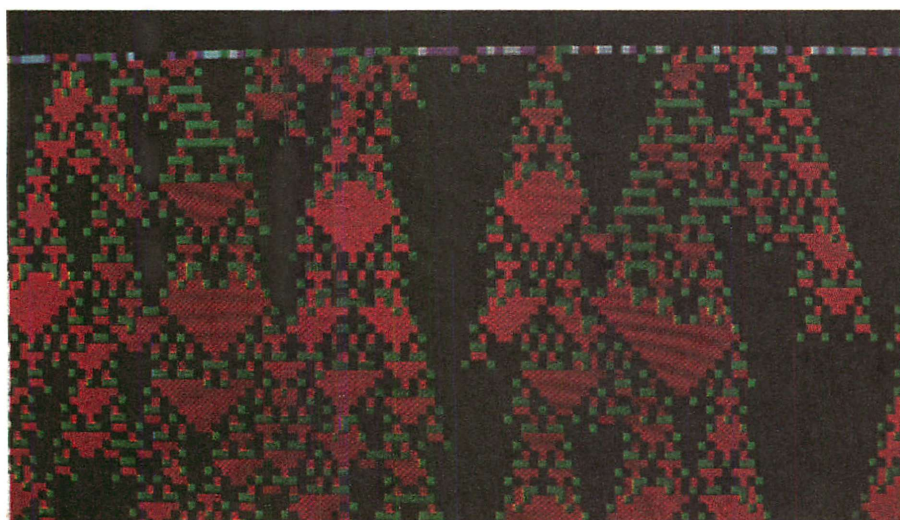
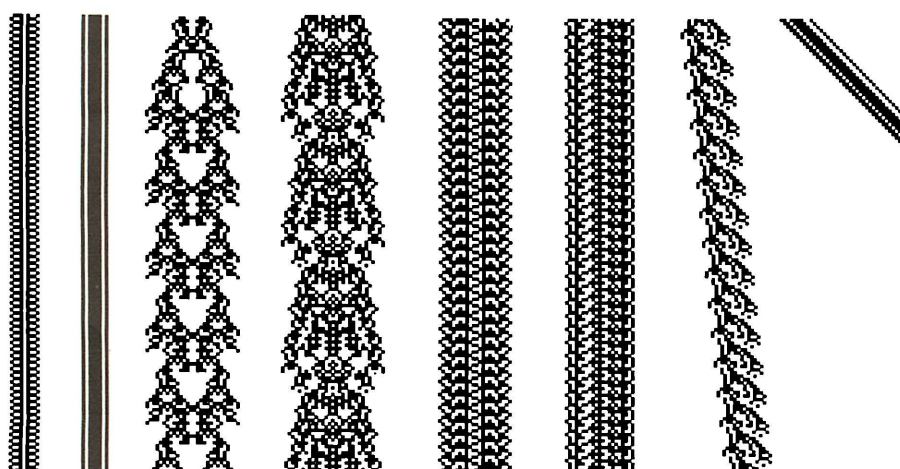
Naturalmente, lo spazio reale, il tempo e molte variabili fisiche sono ritenuti continui anziché discreti (almeno nella scala comunemente presa in considerazione). Non ne consegue, però, che le equazioni differenziali portino di per sé a modelli della natura più validi. Spesso non è il preciso valore numerico di una variabile a essere significativo, ma solo la sua dimensione globale, come il fatto che un particolare punto di un fiocco di neve in crescita sia ghiaccio o vapore acqueo. Gli automi cellulari rendono esplicita questa «discre-

L'evoluzione di un automa cellulare secondo la regola dei due quinti

Algoritmo per automi cellulari basato su regole di transizione «di conto» o «totalistiche»

tezza» su un calcolatore digitale. Inoltre, si può calcolare esattamente la loro evoluzione nel tempo, senza bisogno di approssimazioni. Oltre a ciò, possono fare un uso molto più efficace delle risorse del calcolatore digitale.

Un programma per simulare un automa cellulare può essere scritto anche per i calcolatori più piccoli. Per Bak, del Brookhaven National Laboratory, ha di recente sostenuto in «Physics Today» che molte simulazioni si possono effettuare su un piccolo calcolatore personale più efficacemente e a minor costo che su più potenti macchine usate in partizione di tempo. L'esempio scelto a titolo illustrativo era una simulazione del modello tridimensionale di Ising, fatta con un Commodore VIC 20 e a un costo stimato in 4 dollari.

Il più semplice programma per automa cellulare include semplicemente il metodo che probabilmente si adotterebbe per effettuare la procedura a mano su carta millimetrata. Innanzitutto si stabilisce una matrice di celle, con ogni cella rappresentata da un elemento di memoria del calcolatore. A ogni passo nel tempo, il calcolatore deve occuparsi a turno di ogni cella, esaminare le sue adiacenti e calcolare il valore appropriato del successivo stato della cella. Il calcolo stesso è svolto comodamente cercando il valore in una tabella. Se si considerano solo regole di conto, la tabella richiede soltanto un'entrata per ogni possibile numero di celle «accese». Quando sono ammessi altri tipi di regole, la tabella può diventare piuttosto complicata.

Quando si scrive un programma di questo genere vanno tenuti a mente alcuni accorgimenti. Il più importante è che bisogna evitare di alterare il contenuto di una cella prima che il suo valore sia stato controllato da tutte le altre celle a cui è adiacente. Il modo più semplice per soddisfare questa condizione è conservare due copie della matrice; il programma esamina una copia per stabilire lo stato attuale dell'intorno e immette il risultato del suo calcolo nell'altra copia. Vanno definite anche condizioni di limite. Idealmente la matrice dovrebbe essere infinita, ma la cosa è ovviamente impraticabile. Una tecnica comune consiste nel congiungere i bordi di una matrice, in modo che celle su bordi opposti divengano adiacenti. In una dimensione, una matrice di questo genere è topologicamente un cerchio e in due dimensioni è un toro; pur essendo finita, non ha confini.

Un programma del tipo descritto sopra, che giri su un calcolatore digitale di uso generale, è una procedura sequenziale che simula le azioni di una schiera di molti calcolatori attivi simultaneamente. Molto meglio sarebbe avere davvero una rete di molteplici calcolatori con la struttura di una matrice cellulare. La costruzione di una macchina del genere non è affatto fuori questione: i singoli calcolatori sarebbero così semplici che ce ne starebbero molti su un unico chip di semiconduttore. Anche il fatto che solo calcolatori vicini hanno bisogno di comunicare l'uno con

Il gioco «Vita» evolve sullo schermo della macchina automa cellulare di Tommaso Toffoli

Una configurazione di dendriti è creata da un automa cellulare con una regola di transizione asimmetrica

Le quattro classi di regole totalistiche in una dimensione

Stati successivi di un automa unidimensionale della classe 4

Alcuni componenti di un possibile calcolatore universale

l'altro ridurrebbe la complessità del congegno. Toffoli ha stimato che un simile elaboratore potrebbe operare un milione o forse anche un miliardo di volte più velocemente di un calcolatore di uso generale. Un lavoro preliminare su calcolatori di questo tipo è stato svolto al Massachusetts Institute of Technology e alla Thinking Machines Corporation di Waltham, Massachusetts.

Al posto di un chip specializzato, Toffoli ha costruito una macchina automa cellulare dedicata con normali componenti microelettronici. I calcoli sono effettuati in serie, invece che per tutte le celle in una sola volta, ma, dato che è sintonizzato su un unico tipo di calcolo, il congegno è circa 1000 volte più veloce di un calcolatore di uso generale. La macchina in sé consiste di qualche scheda a circuiti stampati montata su un telaio; è collegata a una unità video a colori ed è controllata da un altro piccolo calcolatore, un Atari 800.

La macchina automa cellulare di Toffoli fornisce una matrice di 256 per 256 celle, ognuna delle quali può avere fino a 256 stati. Lo stato di ogni cella è ricalcolato 60 volte al secondo. Guardare un sistema che si evolve a questo ritmo è molto diverso dal guardare un congegno più lento. Invece di una successione di fotografie statiche si vede un film in movimento. Il gioco Vita non sembra più una maestosa progressione di configurazioni astratte; è più simile all'osservazione attraverso il microscopio di batteri e protozoi che nuotano, ruotano, procreano, mangiano e vengono mangiati.

Un automa cellulare a una dimensione richiede a un calcolatore molte meno risorse, sia spaziali sia temporali, di quante ne richieda un sistema a due dimensioni. Anche scrivere un programma per un sistema a una dimensione è più facile. La matrice lineare ha anche un altro vantaggio rispetto a quella planare: a causa della più semplice struttura geometrica, c'è maggiore possibilità di raggiungere una comprensione analitica dell'evoluzione dell'automa. E questo è proprio quanto ha cercato di fare, negli ultimi anni, Stephen Wolfram dell'Institute for Advanced Study.

Una singola generazione di una matrice a una dimensione è semplicemente una fila di celle, ma si possono tracciare generazioni successive una di seguito all'altra. In questo modo si forma una matrice bidimensionale con un asse dello spazio e un asse del tempo e si può avere a colpo d'occhio l'intera evoluzione del sistema.

Wolfram ha scoperto che tutte le regole di transizione da lui studiate finora si possono inserire in quattro sole classi. La classe 1 è formata da quelle regole la cui evoluzione porta a uno stato stabile e omogeneo; per esempio, tutte le celle potrebbero assumere un valore pari a 1 o a 0. Le regole della classe 2 danno luogo a semplici strutture che possono essere stabili o periodiche, ma che in entrambi i casi rimangono isolate una dall'altra. Le regole della classe 3 creano configurazioni

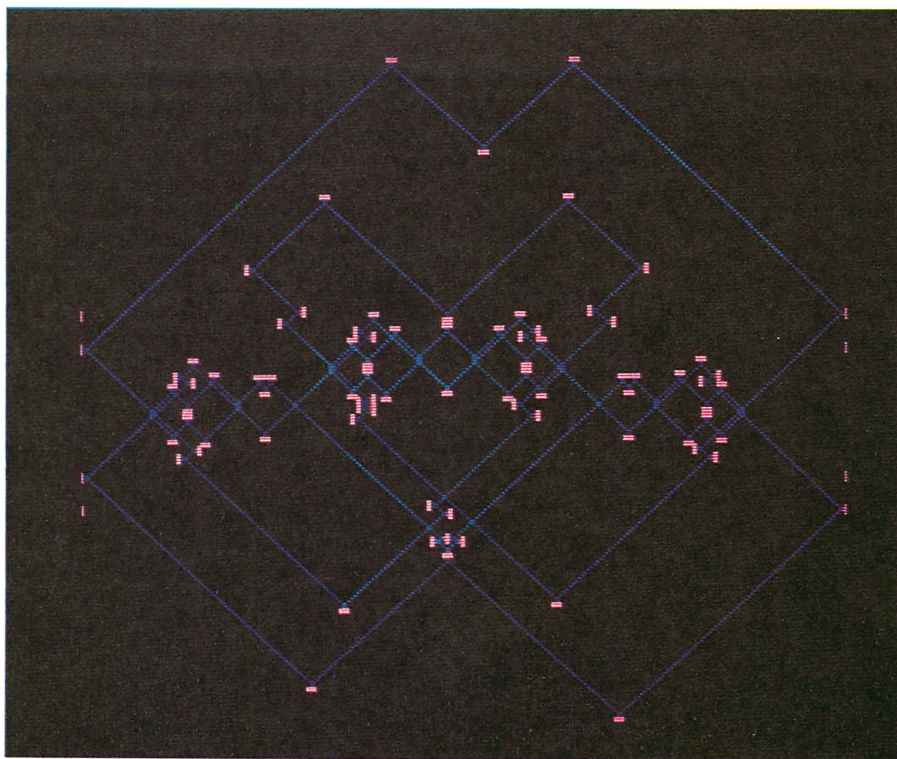
caotiche, anche se non casuali. Nella classe 4 ci sono le poche regole di transizione che generano strutture di sostanziale complessità spaziale e temporale.

Wolfram suppone che gli automi cellulari a una dimensione possono essere i più semplici sistemi ben definiti capaci di un comportamento complesso di autoorganizzazione. In natura molti sistemi dinamici continui hanno questa capacità: evolvono da uno stato casuale di partenza a una struttura altamente ordinata. (Un esempio è il fiocco di neve.) L'evoluzione può essere spiegata in termini di attrattori, che sembrano trascinare il sistema verso un sottoinsieme di tutte le possibili configurazioni.

È stato tracciato un parallelo tra le classi di automi cellulari e i tipi di attrattori che si osservano nei sistemi fisici. Un automa della classe 1 è analogo a un sistema continuo con il più semplice degli attrattori: un punto limite che invariabilmente porta il sistema allo stesso stato finale. L'evoluzione di un automa della classe 2 è invece simile a quella di un sistema con un ciclo limite, un insieme di configurazioni che si ripetono indefinitamente.

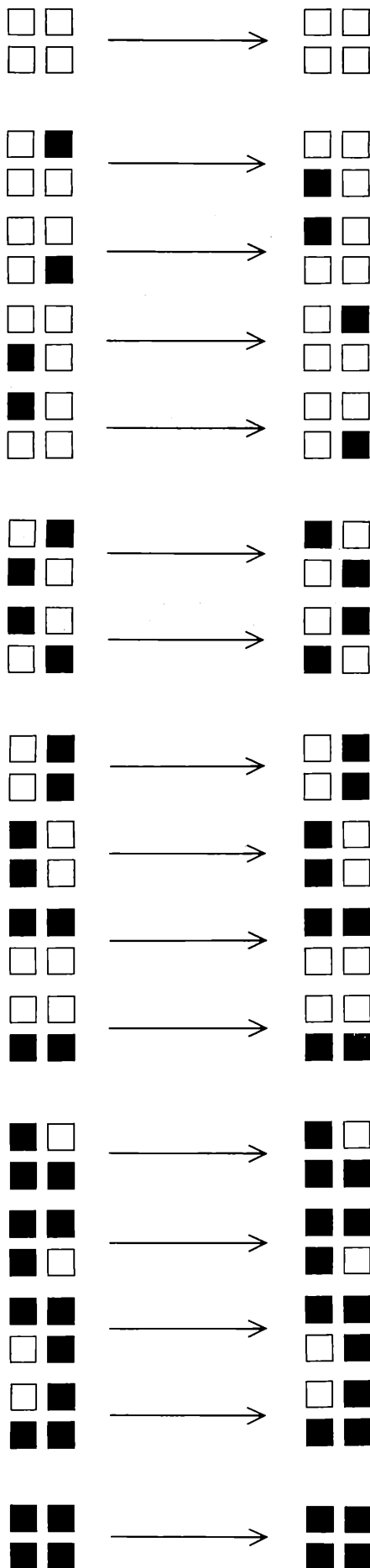
Gli automi della classe 3, con le loro disposizioni caotiche, si possono associare con quelle più interessanti entità dette attrattori strani, caratteristici di fenomeni fisici quali l'inizio di correnti di turbolenza. In un sistema governato da un attrattore strano, l'evoluzione procede verso un sottoinsieme di tutte le configurazioni possibili, ma il sottoinsieme può avere una struttura incredibilmente complicata. Quando si visualizza l'insieme come matrice di punti nello spazio, si tratta in molti casi di un frattale, una figura geometrica con un numero frazionario di dimensioni.

Le distinzioni tra le classi di automi possono essere chiarite prendendo in considerazione un semplice esperimento. Supponiamo di avviare un automa cellulare in qualche configurazione iniziale scelta a caso e di farlo evolvere per molti passi nel tempo; si prenda nota dello stato finale. Si ritorni ora alla configurazione di partenza, si cambi il valore di una singola cella e si faccia evolvere il sistema per lo stesso numero di passi. Che effetto avrà il piccolo cambiamento sullo stato finale? Per un automa della classe 1 non c'è alcuna conseguenza: un sistema della classe 1 raggiunge lo stesso stato finale indipendentemente dallo stato iniziale. Un automa della classe 2 può mostrare qualche effetto, ma limitato a una piccola area vicino al sito in cui è avvenuto il cambiamento. In un sistema della classe 3, invece, l'alterazione di una singola cella può provocare un disturbo che si propaga lungo tutta la matrice.

Le regole della classe 4 sono le più rare e le più interessanti. Alcune funzioni di transizione piuttosto semplici ricadono in questa classe; per esempio, nell'intorno definito in modo da includere la cella centrale e le due celle che le stanno a lato, la regola secondo cui la cella centrale è 1 se due o quattro celle dell'intorno sono 1 porta a configurazioni della classe 4. La sensibilità a piccole variazioni nelle condizioni iniziali è ancora maggiore nella classe 4 che nella classe 3. Si ritiene che per prevedere lo stato futuro di un automa della classe 4 non vi sia nessuna procedura generale più efficace di quella che consiste nel lasciare all'automa stesso il compito di calcolare lo stato.

Il calcolatore a palle di biliardo in azione

Una congettura correlata alla precedente ha una portata ancora più estesa e suggerisce che gli automi della classe 4 si possano considerare dei calcolatori universali. La macchina di Turing è il più familiare tra i dispositivi di questo tipo; se una funzione può essere calcolata, è presumibile che una macchina di Turing possa farlo. Si può dimostrare che altri calcolatori sono universali mostrando la loro equivalenza a una macchina di Turing. Di numerosi automi cellulari a due dimensioni (incluso il gioco Vita) si è dimostrato che sono universali e una dimostrazione è stata fornita anche per un complicato sistema a una dimensione con 18 stati per cella. Gli automi della classe 4 sarebbero i più semplici calcolatori universali conosciuti. Si è individuata la maggior parte dei componenti essenziali. Un importante elemento mancante è un orologio: una struttura che emette una serie di impulsi a intervalli regolari, come il cannone ad alianti nel gioco Vita.

Considerare gli automi cellulari come calcolatori fa pensare che il loro comportamento di autoorganizzazione può essere caratterizzato nei termini delle loro capacità di calcolo. Così, per esempio, insiemi di configurazioni generati dall'evoluzione di un automa cellulare possono essere pensati come un linguaggio formale. Ogni configurazione è considerata una parola del linguaggio, formata da una sequenza di simboli che rappresentano i valori dei siti dell'automa cellulare secondo un insieme di regole grammaticali. Wolfram ha dimostrato che la configurazione generata da qualsiasi automa cellulare dopo un tempo finito può essere descritta da una semplice classe di linguaggi formali noti come linguaggi regolari. Per qualsiasi di questi linguaggi regolari è possibile trovare una semplicissima grammatica. Tale grammatica fornisce una descrizione minimale delle configurazioni dell'automa cellulare e si può assumere la sua dimensione come misura della complessità delle configurazioni. Per automi cellulari delle classi 1 e 2, la complessità tende a un limite finito per tempi lunghi, quindi le strutture generate da quei sistemi sono descritte da linguaggi regolari. Per automi cellulari delle classi 3 e 4, invece, la complessità di solito cresce rapidamente col tempo e appare evidente la necessità di linguaggi formali più complicati per descrivere il comportamento sul lungo periodo di tali sistemi.

Esiste una classe particolare di automi cellulari detti reversibili, o invertibili. Da qualsiasi configurazione di partenza, un automa reversibile che sia fatto evolvere per un qualsiasi numero di passi, poi fermato e fatto girare all'inverso, tornerà esattamente allo stato iniziale. Le configurazioni formate da un automa reversibile tipico hanno un aspetto qualitativamente differente rispetto alle configurazioni caratteristiche di un automa non reversibile. In particolare, se la configurazione iniziale è casuale, essa tende a rimanere casuale: non compare nessuna struttura di autoorganizzazione.

Regola di transizione per un calcolatore a palle di biliardo

Una condizione necessaria per la reversibilità è che la regola di transizione sia deterministica tanto in avanti quanto all'indietro; ogni possibile stato di un intorno, cioè, deve avere sia un unico successore sia un unico predecessore. Il gioco Vita è non reversibile perché il predecessore di uno stato non può essere individuato senza ambiguità: se, per esempio, una cella è attualmente «morta», nella generazione precedente avrebbe potuto avere un numero qualsiasi di adiacenti vivi diverso da tre. Un modo sistematico per creare regole di transizione reversibili è stato inventato da Fredkin e in seguito approfondito da Margolus. L'essenza del metodo sta nel far dipendere lo stato successivo di una cella dai due stati precedenti dell'intorno. Lo stato al momento $t+1$ è dato da una funzione qualsiasi dell'intorno al momento t meno lo stato al momento $t-1$. L'inverso è allora chiaro: lo stato al momento $t-1$ deve essere dato dallo stato al momento t meno lo stato al momento $t+1$.

In ragione della condizione di determinismo bidirezionale, non ci può essere alcun attrattore nell'evoluzione di un automa reversibile. La presenza di un attrattore implica che molti stati iniziali si evolvano lungo percorsi che si fondono l'uno con l'altro; nell'evoluzione rovesciata, i punti di fusione diventerebbero punti di diramazione e il determinismo verrebbe meno. Analogamente, un automa cellulare reversibile non può mai entrare o uscire da un *loop,* ossia un ciclo di stati, perché anche in questo caso si produrrebbe un punto di diramazione in una direzione o nell'altra. Vista l'esclusione di attrattori e delle configurazioni di autoorganizzazione associate, potrebbe sembrare che le regole di transizione reversibili diano luogo ad automi cellulari piuttosto stupidi, ma vi sono in compenso altre caratteristiche che rendono interessanti questi sistemi. Il fatto più notevole è che il contenuto informativo di una configurazione di celle in un automa reversibile è una grandezza che si conserva (non può aumentare o diminuire nel corso dell'evoluzione dell'automa). Questa proprietà rende i sistemi reversibili dei validi modelli di calcolo.

Margolus ha costruito un calcolatore automa cellulare basato su un immaginario sistema meccanico studiato per la prima volta da Fredkin: il modello del calcolo a palle da biliardo. Nel modello, i bit di informazione (1 e 0) sono trasportati da ideali palle da biliardo che si muovono senza attrito e rimbalzano con perfetta elasticità l'una contro l'altra e contro altri ostacoli. La presenza di una palla in una posizione designata rappresenta un 1 binario e l'assenza di una palla rappresenta uno 0 binario. Con dei respingenti disposti in modo opportuno è possibile creare porte logiche analoghe a quelle di un calcolatore elettronico. In una porta AND, per esempio, una palla da biliardo passa attraverso la regione di uscita (e quindi registra un 1 binario) solo se due palle si avvicinano simultaneamente alla porta lungo specifiche traiettorie.

La versione del modello a palle da biliardo proposta da Margolus, basata su un automa cellulare, è un esempio di regola reversibile di transizione semplice ma piuttosto inconsueta. Le celle non sono considerate individualmente ma in blocchi di quattro; ogni possibile configurazione all'interno di un blocco è trasformata in un'unica configurazione prodotto. La regola è fatta in modo che un unico 1 su uno sfondo di 0 si propaghi lungo una delle quattro direzioni diagonali del reticolo a una velocità di una cella per passo; l'1 isolato è l'incarnazione di una palla da biliardo. Un blocco continuo di quattro 1 rimane immutato e agisce come perfetto riflettore. Quando si fa girare il modello sulla macchina automa cellulare di Toffoli, le «palle da biliardo» si muovono velocemente lungo lo schermo in complesse configurazioni intrecciate. Guardando questo movimento ordinato (anche se frenetico), è difficile tenere a mente che il programma non possiede nessuna rappresentazione dei percorsi delle palle ma applica semplicemente un'unica regola a tutte le celle.

Il modello a palle da biliardo e la sua realizzazione con l'automa cellulare hanno un risvolto importante per la teoria del calcolo. È stata avanzata la congettura che qualsiasi calcolatore debba avere componenti che dissipano sia energia sia informazione; secondo questa ipotesi, c'è un limite termodinamico al rendimento di un calcolatore, proprio come c'è un limite al rendimento di una macchina termica. Le perdite di informazione e di energia che si suppongono inevitabili dipendono direttamente dall'irreversibilità del processo di calcolo. (Quando un calcolatore somma i numeri 5 e 3 per ottenere 8, la procedura non può essere rovesciata perché ci sono infiniti numeri che, sommati, avrebbero potuto dare lo stesso risultato.)

Fredkin, Toffoli e Margolus affermano che il modello a palle da biliardo offre un argomento contrario all'idea della dissipazione inevitabile. Nel calcolatore a palle da biliardo non si perde alcuna informazione. In effetti, le stesse palle da biliardo non si possono creare né distruggere e tutta l'informazione che definisce la loro disposizione iniziale viene conservata pur con tutto l'evolversi del sistema. Gli ingressi per un'operazione di addizione possono essere recuperati semplicemente invertendo le traiettorie. In linea di principio, il calcolatore a palle da biliardo potrebbe operare senza alcun consumo interno di energia.

Il collegamento tra fisica e calcolo automatico è stato tracciato con particolare chiarezza da Toffoli in una affermazione che si potrebbe leggere come una descrizione del più grande degli automi cellulari. «In un certo senso - scrive - la natura ha continuato a calcolare lo "stato successivo" dell'universo per miliardi di anni; tutto quello che dobbiamo fare - e, in effetti, tutto quello che *possiamo* fare - è "saltare in groppa" a questo enorme calcolo in atto e cercare di scoprire quali parti di esso si avvicinano a dove noi vogliamo andare.»

Un calcolatore trappola per l'alacre castoro, la più attiva fra le macchine di Turing

di A. K. Dewdney

Le Scienze, ottobre 1984

A eccezione forse delle api, i castori sono gli animali più attivi che esistano: tutto il giorno si affaccendano nelle quiete acque del Nord, portando rami verso le loro dighe. Fu sicuramente questo comportamento a indurre Tibor Rado, della Ohio State University, a dare il nome di «gioco dell'alacre castoro» a un problema relativo alla macchina di Turing. Agli inizi degli anni sessanta, Rado si chiedeva quanti 1 si potessero far stampare a una macchina di Turing prima che questa si fermasse. Più precisamente, se una macchina di Turing a n possibili stati inizia a funzionare su un nastro che contiene solo zeri, qual è il massimo numero di 1 che può stampare sul nastro prima di fermarsi? La risposta è nota per $n = 1$, $n = 2$, $n = 3$ e $n = 4$, ma non per $n = 5$, o per qualsiasi valore di n superiore a 5.

Nel 1983 si è svolta a Dortmund, nella Germania Occidentale, una gara per vedere chi scopriva il più attivo «castoro» a cinque stati. Nell'anno precedente la gara erano stati scritti programmi per generare macchine di Turing adatte ed era stato sviluppato l'hardware per sottoporre a verifica tali macchine. Nel corso di questo lavoro furono scoperti parecchi castori dallo strano comportamento e il genere *Castor* dovette essere allargato per includervi varie specie fino ad allora ignote agli zoologi.

La natura della macchina di Turing e il suo posto nella scienza dei calcolatori, sono stati analizzati da John E. Hopcroft, della Cornell University, nell'articolo *Macchine di Turing* apparso sulla rivista «Le Scienze» n. 191, nel luglio 1984. Una macchina di Turing è fatta di un nastro infinito, una testina per leggere e scrivere simboli sul nastro e un'unità di controllo con un numero finito di stati interni (*si veda l'illustrazione della pagina successiva*). Queste componenti possono essere viste come l'*hardware* del congegno, mentre il contenuto dell'unità di controllo è il *software*, il programma della macchina di Turing. È il programma che distingue una macchina di Turing da un'altra. Il programma è una tabella che la macchina consulta per stabilire qual è la successiva azione da compiere. Per ogni possibile stato dell'unità di controllo e per ogni possibile simbolo in una certa posizione della testina, un'entrata della tabella dice alla macchina quale simbolo stampare sul nastro, in quale direzione muovere la testina e che stato passare ad assumere. Nel nostro caso, tutte le macchine di Turing esaminate iniziano nello stato 1.

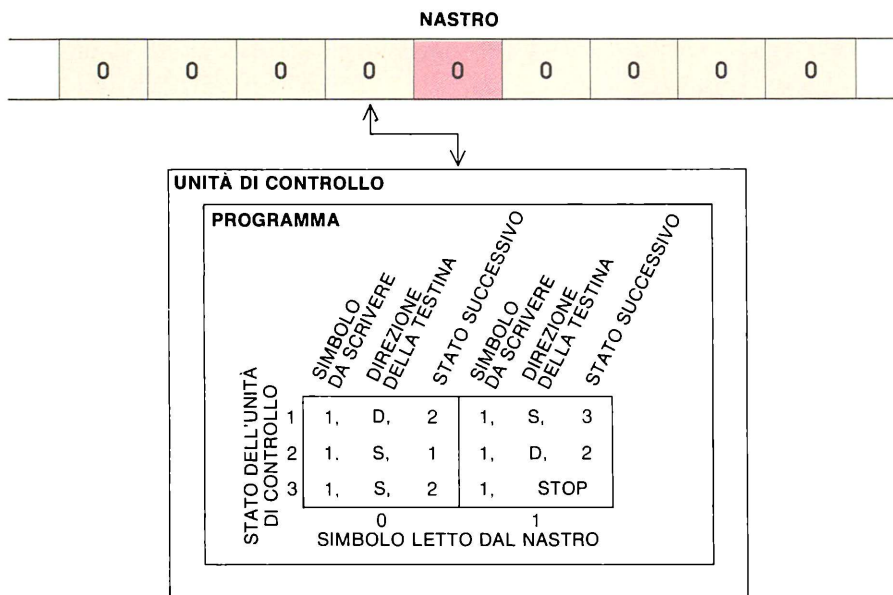
Si possono tracciare le azioni di una macchina di Turing scrivendo lo stato dell'unità di controllo e i simboli segnati sul nastro (o una sua regione) in momenti successivi; inoltre si dovrebbe indicare quale sia la casella del nastro attualmente in esame. L'illustrazione in basso della pagina successiva è una traccia della macchina di Turing descritta nella stessa pagina. Ogni linea della successione è una «descrizione istantanea» della macchina. La descrizione è differente da quella adottata da Hopcroft, ma l'informazione è la stessa. Contrariamente alle convenzioni descritte nell'articolo di Hopcroft citato, ho anche reso infinito il nastro in entrambe le direzioni e ho ammesso che un simbolo venga stampato nel corso della transizione finale della macchina (quando entra nello stato di fermo), differenze che comunque non incidono su ciò che una macchina di Turing può o non può fare. Il formato che ho scelto per la descrizione istantanea è compatibile con quello usato nella gara dei programmi per il problema dell'alacre castoro.

Un alacre castoro a n stati è una macchina di Turing a n stati che soddisfa due condizioni: la prima è che quando viene fatta partire su un nastro pieno di 0, alla fine si ferma; la seconda è che scrive almeno tanti 1 quanti ne scrive qualsiasi altra macchina a n stati che si fermi. Nell'illustrazione in alto a pagina 87 si vedono alacri castori a uno e tre stati. Ogni macchina di Turing è rappresentata da un diagramma di transizione di stato, in cui uno stato è rappresentato da un cerchio numerato e una transizione tra stati da una freccia. Le etichette sulle frecce descrivono l'azione della macchina di Turing. Per esempio supponiamo che l'alacre castoro a tre stati sia nello stato 1 e legga uno 0 sul nastro. La freccia seguita in queste circostanze porta l'indicazione «0, 1, *D*» e conduce allo stato 2. Quindi la macchina, avendo letto uno 0, scrive un 1 sul nastro, sposta la testina di una casella verso destra ed entra nello stato 2.

Il numero massimo di 1 che può essere prodotto da una macchina di Turing a n stati che si fermi è indicato con $\Sigma(n)$. Come detto prima, il valore di $\Sigma(n)$ è noto solo per i primi quattro valori di n. L'alacre castoro a uno stato scrive un unico 1 prima di fermarsi; in altre parole, $\Sigma(1)$ è uguale a 1. Un alacre castoro a due stati produce una sequenza di quattro 1. I lettori sono in grado di escogitare una macchina del genere? Un alacre castoro a tre stati scrive sei 1; nelle illustrazioni della pagina successiva si possono vedere il programma e la sequenza di descrizioni istantanee di un castoro a tre stati, mentre a pagina 87 è raffigurato il suo diagramma di transizione di stato. Il castoro a tre stati fu scoperto nel 1962 da Rado e da Shen Lin degli AT&T Bell Laboratories. Nel 1973, Bruno Weimann dell'Università di Bonn scoprì un alacre castoro a quattro stati, che dà come uscita 13 1 consecutivi. Da allora i teorici sono alla ricerca di un alacre castoro a cinque stati.

La sfida degli alacri castori fu organizzata da Frank Wankmuller e si tenne nel gennaio 1983 all'Università di Dortmund durante un congresso di scienza teorica del calcolatore. Vennero iscritte alla gara 133 macchine di Turing a cinque stati e vinse Uwe Schult, di Amburgo, con una macchina che produsse 501 1 prima di fermarsi. Nell'illustrazione in basso a pagina 87 si può osservare un diagramma di transizione di stato della macchina vincente. Secondo classificato fu Jochen Ludewig, del Centro di ricerche Brown Boveri di Baden, con una macchina di Turing che stampò 240 1.

La macchina di Turing ideata da Schult è un alacre castoro? Schult, d'accordo con Wankmuller e Ludewig, ritiene di sì. In altri termini, pensa che nessuna macchina di Turing a cinque stati possa produrre più di 501 1 prima di fermarsi. Come si potrebbe dimostrare una simile affermazione? La risposta sta in una ricerca esaustiva compiuta dal calcolatore, una ricerca sul tipo di quella condotta da Schult per identificare la sua macchina di Turing campione. Prima di descrivere il tentativo di Schult di catturare l'alacre castoro a cinque stati nel suo calcolatore, vorrei considerare più da vicino la funzione $\Sigma(n)$ per cercare di capire perché il gioco dell'alacre castoro sia così difficile, anche con l'aiuto di un calcolatore.

La funzione $\Sigma(n)$ ha una proprietà straordinaria: non è computabile. Cresce troppo rapidamente. Dai primi quattro valori di $\Sigma(n)$ - vale a dire 1, 4, 6 e 13 - potrebbe sembrare che il tasso di crescita sia modesto. Se 501 è davvero il numero massimo di 1 per una macchina a cinque stati, l'incremento di $\Sigma(n)$ ancora non sembrerebbe più veloce di una funzione esponenziale. Schult ha trovato una macchina di Turing a sei stati che produce 2075 1, il che fa pensare ancora a un tasso di crescita abbastanza trattabile. D'altra parte, Schult ha anche trovato una macchina a 12 stati che genera talmente tanti 1 che il numero deve essere espresso da una formula da far venire le vertigini:

Una macchina di Turing e il suo programma

STATO	NASTRO								
1	0	0	0	0	0	**0**	0	0	0
2	0	0	0	0	1	0	**0**	0	0
1	0	0	0	0	1	**1**	0	0	0
3	0	0	0	0	**1**	1	0	0	0
2	0	0	0	**1**	1	1	0	0	0
1	0	0	**1**	1	1	1	0	0	0
2	0	1	1	**1**	1	1	0	0	0
2	0	1	1	1	**1**	1	0	0	0
2	0	1	1	1	1	**1**	0	0	0
2	0	1	1	1	1	1	**0**	0	0
2	0	1	1	1	1	1	0	**0**	0
1	0	1	1	1	1	1	**1**	0	0
3	0	1	1	1	1	**1**	1	0	0
STOP	0	1	1	1	1	1	1	0	0

«Descrizioni istantanee» che tracciano il funzionamento della macchina di Turing

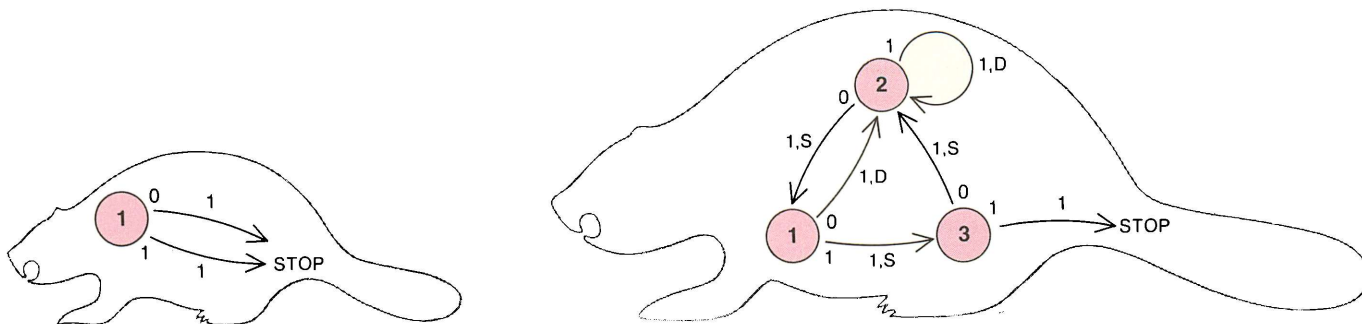
$$6 \times 4096^{4096^{4096^{\cdot^{\cdot^{\cdot^{4096^{4}}}}}}}$$

Il numero 4096 compare nella formula 166 volte, di cui 162 nella «zona crepuscolare» rappresentata dai tre puntini. La formula può essere valutata dall'alto verso il basso: innanzitutto si eleva 4096 alla quarta potenza, poi si eleva 4096 alla potenza del numero risultante, poi si eleva 4096 alla potenza di *quel* numero, e così via. Quando si è raggiunto il fondo, si moltiplica per 6.

Chi non sente girare la testa di fronte a una successione di 1 così lunga è invitato a costruire un numero ancora più grande. Scrivete una formula a piacere in cui dei numeri siano moltiplicati o elevati a potenza; volendo potete anche sostituire n ai numeri. Qualsiasi formula riusciate a escogitare, per qualche valore di n abbastanza grande l'alacre castoro a n stati produrrà più 1 di quelli specificati dalla formula. Ne segue che $\Sigma(n)$ non può essere calcolata per valori arbitrariamente grandi di n. Il meglio che possiate fare è calcolare $\Sigma(n)$ per qualche piccolo valore prefissato di n.

Non è certo sorprendente che il gioco dell'alacre castoro sia per lo più giocato con l'aiuto di un calcolatore. Il metodo fondamentale consiste nell'esaminare sistematicamente tutte le macchine di Turing a n stati. Ogni volta che si genera una nuova macchina, viene simulato il suo comportamento su un nastro pieno di 0. Se la macchina si ferma dopo non più di un dato numero di passi, il numero di 1 stampati viene confrontato con il risultato della «più alacre» macchina di Turing trovata fino a quel momento. Di tanto in tanto si scopre un nuovo campione.

Questo metodo di ricerca dell'alacre castoro a n stati ha due grosse falle. Primo, il numero di macchine di Turing da generare è immenso; per esempio, ci sono 63 403 380 965 376 macchine a cinque stati. Secondo, non si sa quanto a lungo aspettare che una macchina si fermi; il numero massimo di transizioni che una macchina a n stati può effettuare (per poi finalmente fermarsi), una funzione denotata $s(n)$, è a sua volta un numero non computabile. Ovviamente, $s(n)$ cresce ancora più velocemente di $\Sigma(n)$, dato che una macchina di Turing deve compiere una transizione di stato ogni volta che stampa un 1. Come rilevato da Hopcroft, calcolare $s(n)$ equivale a risolvere il problema della fermata per le macchine di Turing, uno dei primi problemi che Turing dimostrò essere indecidibili.

Nel 1982, Schult trasformò il suo calcolatore personale Apple II in una trappola per alacri castori. Ampliò l'originario elaboratore centrale del calcolatore con una scheda dotata di un microelaboratore Motorola 6809 e scrisse il suo programma

Alacri castori a uno e tre stati

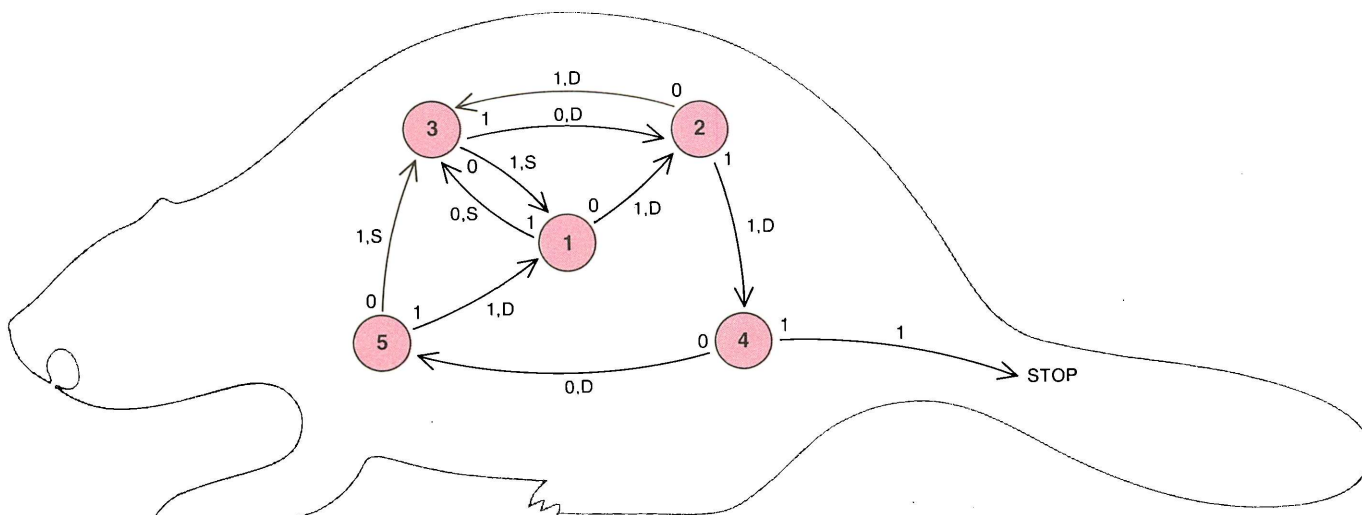
di ricerca nel linguaggio macchina dell'elaboratore ausiliario. Per sottoporre a verifica l'enorme numero di macchine di Turing generate dal programma, Schult costruì una vera e propria macchina di Turing, con un hardware fatto di componenti elettronici standard montati su un'altra scheda inserita in un connettore d'espansione dell'Apple II. Il dispositivo fornisce un nastro simulato di 4096 caselle e inoltre registri per immagazzinare il programma, lo stato attuale e l'attuale posizione della testina della macchina di Turing. Secondo le stime di Schult, senza un hardware così specializzato la sua ricerca avrebbe coperto 20 mesi di funzionamento del calcolatore. Anche con le espansioni di hardware, comunque, l'Apple II impiegò 803 ore per trovare la macchina di Turing vincente.

Progettando il software necessario, Schult ottenne ulteriori vantaggi facendo sì che il programma di ricerca e l'hardware della macchina di Turing interagissero strettamente. Il programma completava sistematicamente, in tutti i modi possibili, la tabella di transizione per una macchina di Turing a cinque stati. Prima ancora che fosse completata, la tabella veniva sottoposta all'hardware della macchina di Turing per la verifica. In molti casi si trovava che una tabella incompleta specificava una macchina che usciva dai limiti di tempo o spazio prima di raggiungere una delle entrate non definite. La tabella incompleta e tutti i suoi possibili completamenti potevano così essere accantonati.

Sebbene Schult abbia superato in larga misura il problema di controllare grandi numeri di macchine di Turing, il suo modo di affrontare il problema della fermata non è, per così dire, a prova di bomba. In mancanza di informazioni precise su $s(5)$ - il numero massimo di transizioni effettuabili da una macchina di Turing a cinque stati che si fermi - si può solo cercare di indovinarne il valore. Schult scelse il limite di 500 000; in altre parole, adottò l'ipotesi di lavoro che se una macchina non si fosse fermata dopo 500 000 transizioni non si sarebbe mai fermata. Per necessità, impose anche limitazioni di spazio ai suoi possibili alacri castori; dato che il nastro simulato aveva solo 4096 caselle e dato che le macchine di Turing partivano sempre al centro di questo nastro finito, un candidato era considerato un «corridore» se si spostava di più di 2048 caselle dalla posizione iniziale. Un corridore è una macchina di Turing che non solo non si ferma ma continua indefinitamente a visitare nuove caselle del nastro.

Delle 133 macchine di Turing iscritte alla gara di Dortmund, solo quattro produssero più di 100 1. Il funzionamento di ogni macchina di Turing era simulato con un calcolatore Siemens 7.748. Ci volle più di un'ora di elaborazione per determinare il vincitore.

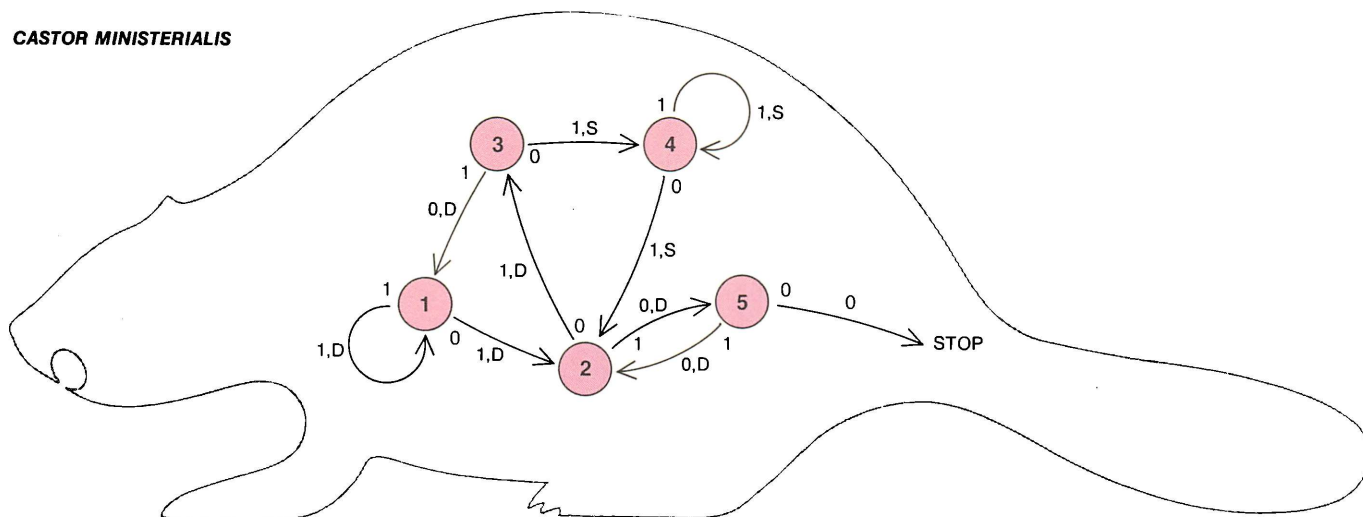
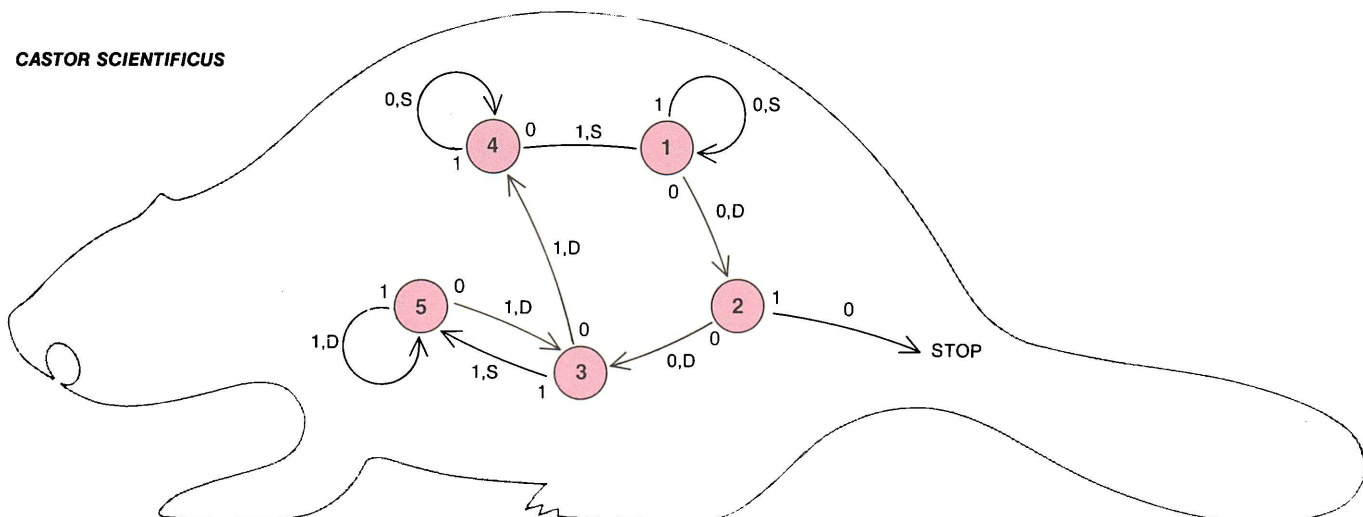
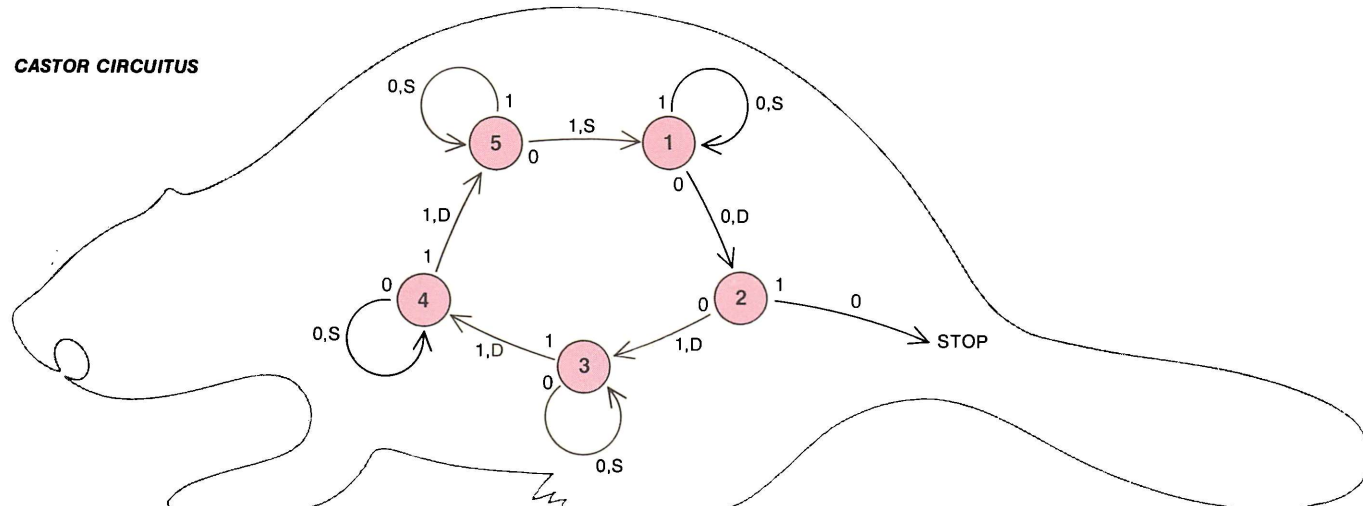
Ludewig, il secondo classificato, scrisse il suo programma di ricerca dell'alacre castoro nel linguaggio di programmazione Pascal e lo fece girare su un potente minicalcolatore, il VAX, costruito dalla Digital Equipment Corporation. Nonostante una maggiore raffinatezza nell'analisi delle macchine di Turing candidate, furono necessarie 1647 ore di tempo dell'elaboratore centrale per scoprire il suo campione, la macchina di Turing che produceva 240 1. Cosa non sorprendente, anche Schult trovò la macchina di Ludewig ed è altrettanto interessante il fatto che egli non trovò alcuna macchina tra quella di Ludewig e la sua. A quanto pare, una macchina di Turing a cinque stati che si fermi e che stampi più di 240 1 ne deve stampare almeno 501.

Ludewig, nel corso delle sue ricerche, scoprì un certo numero di strane macchine di Turing dal comportamento simile a quello del castoro. Un castoro ha altri modi di tenersi occupato oltre a quello di stampare degli 1. Per esempio, senza stampare molti 1 una macchina di Turing può coprire una notevole distanza dalla sua casella di partenza e poi fermarsi. In alternativa, senza stampare molti 1 o anche senza spostarsi di molto, può passare attraverso una gran numero di transizioni

Il candidato di Uwe Schult per un alacre castoro a cinque stati

prima di fermarsi. Tra le macchine messe alla prova a Dortmund, quella di Schult vinse in tutte e tre le categorie. D'altra parte, Ludewig scoprì tre castori che non stampano nessun 1 ma in compenso esplorano un ampio territorio oppure perdono molto tempo in attività infruttuose *(si veda l'illustrazione qui in basso)*. Di conseguenza è stato dato un nome a tre nuove specie di castori:

Castor ministerialis (nome comune, castoro impiegato). Questa intraprendente creatura cerca di ottenere il massimo avanzamento possibile senza produrre nulla. L'esemplare tipo è un castoro a cinque stati che non produce nessun 1 e si sposta di 11 caselle dalla posizione di partenza.

Castor scientificus (nome comune, castoro scienziato). Questo animale, che come il precedente non produce nulla, cerca di massimizzare l'attività globale, forse nel tentativo di assicurarsi finanziamenti. Si è potuto osservare un membro a cinque stati di questa specie che compiva 187 transizioni senza scrivere un solo 1.

Castor circuitus (nome comune, castoro circuito). Il castoro circuito non produce nulla e non va da nessuna parte,

Tre nuove specie di castori che dopo una grande attività non lasciano nessun 1 sul nastro

ma nel processo genera una massima quantità di attività. Come suggerisce il diagramma di transizione di stato, tende a passare una gran parte del suo tempo nel girare a vuoto. Il più attivo esemplare a cinque stati trovato finora compie 67 transizioni prima di fermarsi esattamente là dov'era partito.

Sarebbe interessante vedere degli esemplari a tre stati di questi strani castori. Qualsiasi tentativo di trovarli sarebbe certamente favorito dall'uso del calcolatore (personale o altro), anche solo per controllare i programmi per macchina di Turing ideati a mente.

È facile scrivere un simulatore per macchina di Turing. Per rappresentare il nastro si usa una matrice a una dimensione; il contenuto della matrice, fatto solo di 0 e 1, può essere mostrato su video. Si ottiene il massimo di informazione dal video se è indicata la posizione della testina. Per esempio, lo stato attuale della macchina potrebbe essere visualizzato direttamente sotto il simbolo in esame.

Per rappresentare il programma della macchina di Turing è necessaria una matrice bidimensionale. Ogni elemento della matrice è un insieme di istruzioni per la macchina; devono essere fornite istruzioni per ogni stato dell'unità di controllo e per ogni possibile simbolo del nastro. Per una macchina di Turing a tre stati la matrice ha tre righe e due colonne e la sua struttura è esattamente quella del programma che si vede nell'illustrazione in alto a pagina 86. Lo stato della macchina specifica una riga della matrice e il simbolo sotto la testina del nastro specifica una colonna; le istruzioni che si trovano all'intersezione della riga e della colonna designate definiscono la successiva azione della macchina di Turing.

Supponiamo che la macchina sia nello stato 1 e che il simbolo sul nastro sia uno 0. Consultando la riga 1 e la colonna 0 della matrice, il simulatore trova le istruzioni «1,*D*,2». La macchina, quindi, deve scrivere un 1 sul nastro, muovere la testina a destra di una casella e assumere lo stato 2. Un modo per realizzare queste istruzioni consiste nel definire tre variabili, diciamo STATO, TESTINA e SIMBOLO. All'inizio di un ciclo, i valori di STATO e SIMBOLO determinano in che punto della tabella la macchina cerca le successive istruzioni. Il primo componente dell'istruzione trovata (in questo caso un 1) viene scritto sul nastro; il secondo componente (*D*) diviene il nuovo valore di TESTINA e il terzo componente (2) diviene il valore di STATO. La testina viene poi spostata (nella direzione indicata dal valore di TESTINA) e il simbolo trovato nella nuova posizione diviene il valore di SIMBOLO. Il ciclo quindi riparte nuovamente.

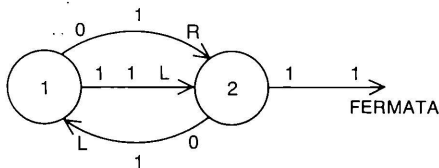
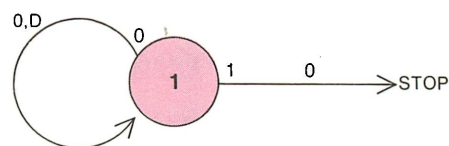
Si possono adottare varie strategie per rendere più semplice e più efficace la programmazione di questo genere di schema. Per esempio, le lettere *S* e *D* possono essere sostituite da numeri, che di solito sono più facili da trattare nel calcolatore. Inoltre, la transizione che porta allo stato di fermo richiede un trattamento particolare nel programma.

Potreste utilizzare un simulatore di macchina di Turing per controllare le vostre risposte ai seguenti piccoli rompicapo, ma non è assolutamente necessario alla loro soluzione.

Immaginate di avere acquistato, nel negozio sotto casa, una partita di nastri usati per macchina di Turing. Prima di sguinzagliarvi sopra il vostro alacre castoro, i nastri devono essere ripuliti: tutti gli 1 devono essere riportati a 0. Invece di pulire voi stessi i nastri, decidete di ideare una semplice macchina di Turing che faccia il lavoro per voi.

Uno dei nastri contiene tutti 0 tranne un unico 1. Dovete creare una macchina di Turing che trovi l'1, lo cancelli (trasformandolo in uno 0) e poi si fermi. Naturalmente, meno stati ha la vostra macchina pulisci-nastro, più elegante sarà. Il pulisci-nastro dell'illustrazione di questa pagina è estremamente elegante. Sfortunatamente, lavora solo metà del tempo!

Gli altri nastri sono come il primo tranne per il fatto che hanno più 1; in ogni caso, però, si sa che il numero di 1 è finito. Siete in grado di costruire un pulisci-nastro che riporti a 0 tutti gli 1? Naturalmente, non si fermerà mai.

I tre rompicapo sull'alacre castoro sono stati risolti da Martin J. Maney di Palatine, Illinois. Qui di seguito è illustrato il suo alacre castoro a due stati. Partendo con un nastro vuoto, produce quattro 1 prima di fermarsi.

È meglio riassumere a parole le soluzioni di Maney. Una macchina di Turing che cancelli l'unico 1 di un nastro utilizza due 1 come contrassegni. A ogni stadio, fa la spola da un 1 all'altro, controllando se subito al di là di ciascuno di essi c'è l'1 da cancellare. Se così è, la macchina cancella tutti e tre gli 1 e si ferma. In caso contrario, la macchina sposta di una casella verso l'esterno il contrassegno e torna all'altro contrassegno. Il pulisci-nastro per nastri che contengano più 1 funziona analogamente ma non si può mai fermare. Come potrebbe farlo, dal momento che qualche regione del nastro non ancora esplorata potrebbe contenere un 1? Alacri castori a due stati sono stati trovati da Peter J. Marineau di Troy, New York, e

Una macchina pulisci-nastro

Dave Kaplan di Deer Park, New York. Marineau ha anche risolto il problema del pulisci-nastro, descrivendo i suoi 1 mobili come «scope» che spazzano il nastro.

Raphael M. Robinson di Berkeley, California, ha scritto un programma di simulazione di una macchina di Turing per il suo IBM PC. Osservando l'alacre castoro ideato da Uwe Schult mentre scriveva i suoi 501 1, Robinson ha notato che prima di fermarsi esso produceva uno schema ricorrente e sempre più lungo di 0 e 1 alternati. Partendo da un nastro vuoto, le lunghezze successive di questo schema erano 0, 6, 13, 28, 48, 78, 121, 190, 289, 442 e 667. L'ultimo schema conteneva 501 1. Robinson ha provato a studiare il comportamento della macchina di Schult partendo non con un nastro vuoto ma con uno degli schemi alternanti. Iniziando con uno schema di lunghezza 9 (contenente cinque 1), la macchina si è fermata dopo 12 870 233 passi, dopo aver prodotto un nuovo schema contenente 4911 1. Ci voleva uno spazio tre volte superiore e un numero di passi 25 volte superiore rispetto alla macchina di Schult fatta partire su un nastro vuoto. Che un cambiamento così modesto nel nastro d'ingresso debba produrre un comportamento così stravagante lascia perplesso Robinson, il quale scrive: «Mi sembra che questi risultati gettino seri dubbi sulle restrizioni di spazio e di tempo fissate da Schult».

Sembra che Bruno Weimann, dell'Università di Bonn, non sia stato il primo a scoprire un alacre castoro a quattro stati. Allen H. Brady, attualmente all'Università del Nevada a Reno, ne scoprì uno dieci anni prima di Weimann. A quel tempo, Brady lavorava alla Oregon State University; mascotte della scuola è il castoro e il calcolatore usato da Brady per le sue ricerche si trovava nei pressi di Beaverton (*beaver* = castoro). Brady condivide lo scetticismo di Robinson. «So, dalla soluzione del problema per quattro stati, che la soluzione del problema per cinque stati è lungi dall'essere decisa. Il punto centrale è decidere che ogni macchina in presunta "fuga" non si fermerà effettivamente mai... Dato che le macchine diventano sempre più complesse, questa decisione diventerà sempre più difficile, finendo col racchiudere in sé profondissimi problemi matematici irrisolti... I problemi della fermata a partire da un nastro vuoto relativi a singole macchine diventeranno terreni matematici unicamente individuali.»

Costruire calcolatori a una sola dimensione getta luce su fenomeni irriducibilmente complessi

di A. K. Dewdney

Le Scienze, luglio 1985

Immersi come siamo oggi in un mondo di calcolatori artificiali, è interessante considerare la possibilità di essere circondati anche da calcolatori naturali. Forse calcolatori fatti di acqua, vento e legno (tanto per citare solo qualche possibilità) gorgogliano, sibilano o crescono dolcemente senza che sospettiamo l'equivalenza di queste attività con un ribollire di calcolo che costituisce la migliore descrizione di se stesso. Questo non significa che tali sistemi naturali calcolino in modo convenzionale, ma solo che la loro struttura rende il calcolo una possibilità latente.

Un sostenitore eloquente di questa idea è Stephen Wolfram, fisico teorico dell'Institute for Advanced Study di Princeton. Wolfram rileva che un flusso turbolento di fluido o la crescita di una pianta sono fatti di componenti piuttosto semplici il cui comportamento combinato è tanto complesso da non essere riducibile a un enunciato matematico - esso stesso costituisce la propria migliore descrizione. L'irriducibilità di un sistema naturale deriverebbe da una dimostrazione della sua capacità di immagazzinare, trasmettere e manipolare informazione; cioè dalla sua capacità di calcolare.

Wolfram ha descritto l'uso di automi cellulari per lo studio di questa possibilità, proponendo di trovare un automa cellulare che calcoli e al contempo imiti un sistema naturale. La ricerca di Wolfram si incentra sui più semplici fra tutti i possibili automi cellulari, quelli a una sola dimensione.

Questi automi sono costituiti da elementi semplici che combinandosi producono complessità. Wolfram ipotizza che nascosti tra loro vi siano veri calcolatori, grandi matrici lineari di cellule che passano da stato a stato e possono eseguire qualsiasi calcolo di cui sia capace un calcolatore tridimensionale. Wolfram, che è attualmente impegnato a cercare tra miriadi di automi cellulari, non disdegna eventuali aiuti di dilettanti in questa audace e raffinata impresa. Più avanti parlerò dettagliatamente della ricerca e delle sue conseguenze per i calcolatori naturali.

Prima di imbarcarsi in quell'avventura, i lettori sono invitati a un breve viaggio (in termini computazionali) dalla terra delle tre dimensioni fino ai confini incredibilmente ristretti della singola dimensione. Un buon punto di partenza sono i calcolatori a tre dimensioni, quelli che oggi popolano gli uffici, le aziende e le nostre case. Essi sono fatti di elementi abbastanza semplici collegati in modo complesso. Non parlo qui dei dispositivi di ingresso e di uscita, ma del cuore della macchina, una sottile piastrina di silicio che ospita migliaia di porte logiche, elementi di memoria, registri e altri componenti, tutti collegati da una elegante distribuzione di sottili conduttori. Il fatto che i circuiti aderiscano a una superficie di silicio non significa che siano a due dimensioni. Innanzitutto, quando due collegamenti si incrociano uno deve passare sotto l'altro. Inoltre, il substrato di silicio dei circuiti ha un ruolo di mediazione per la funzione di ogni componente logico.

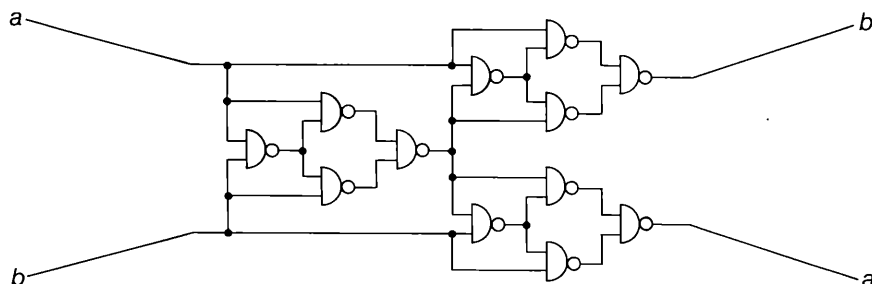
Calcolatori a due dimensioni si possono trovare solo in spazi bidimensionali, come il Pianiverso. Questo regno è abitato da una popolazione di esseri chiamati Ardeani, i quali sono riusciti a costruire un calcolatore a due dimensioni utilizzando un solo tipo di elemento logico, che chiamiamo porta NAND e la cui uscita è un 1 se almeno uno dei suoi ingressi è uno 0. Non solo si può costruire un calcolatore partendo unicamente da queste porte, ma si può anche risolvere lo spinoso problema dei collegamenti che si incrociano. Gli Ardeani creano uno speciale circuito piano con 12 porte NAND fatto in modo che quando due segnali entrano nel circuito da sinistra nell'ordine *ab* lo lasciano da destra nell'ordine *ba (si veda l'illustrazione di questa pagina)*. Due segnali, quindi, possono incrociarsi anche se i collegamenti non possono. Il numero 12, però, sembra un tantino eccessivo e gli Ardeani sarebbero grati al lettore che riuscisse a trovare un circuito NAND più semplice che fosse ancora in grado di far incrociare segnali.

C'è anche uno spazio bidimensionale discreto chiamato Vita, il gioco inventato da John Horton Conway. Brian Hayes ha spiegato come realizzare il gioco con un programma di tabellone elettronico. Vita è un reticolo bidimensionale infinito di cellule quadrate i cui stati sono influenzati dagli stati di cellule vicine. Anche il tempo è discreto e da un battito di un orologio cosmico all'altro lo stato di vita o di morte di ogni cellula dipende da un semplicissimo insieme di regole:

Se una cellula è morta all'istante t, diviene viva all'istante $t + 1$ se e solo se esattamente tre dei suoi otto vicini sono vivi all'istante t.

Se una cellula è viva all'istante t, muore all'istante $t + 1$ se e solo se meno di due o più di tre vicini sono vivi all'istante t.

Con questo insieme di regole operanti su tutto il reticolo di Vita, una configurazione iniziale di cellule vive può continuare a crescere, entrare in uno schema ciclico o infine morire. Dopo più di un decennio di entusiastiche sperimentazioni, è apparso chiaro che Vita è molto più complesso di quanto si fosse pensato.

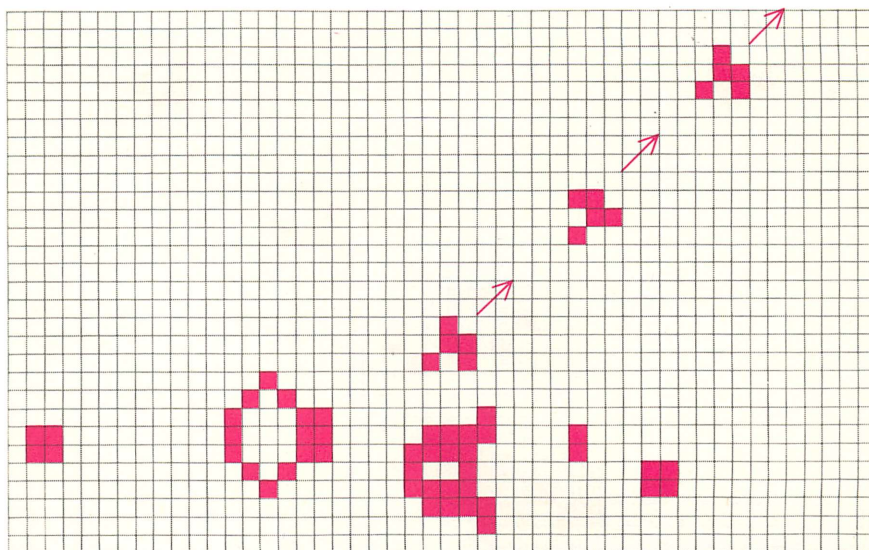
Innanzitutto, è risultato che si possono costruire calcolatori nello spazio cellulare di Vita. Questa scoperta è avvenuta gradualmente nel corso di alcuni anni. Nel 1969, poco dopo aver inventato il gioco, Conway scoprì una piccola e curiosa configurazione che oggi viene chiamata aliante e che attraversava quattro generazioni per poi riprendere la sua forma originale, ma spostata di una cellula lungo la diagonale. Su uno schermo che mostri l'uscita di un programma Vita particolarmente veloce, un aliante sembra una piccola e fantastica creatura che dimeni la coda mentre scorre lungo lo schermo a una velocità pari a

Come far incrociare segnali in un calcolatore a due dimensioni

un quarto di quella della luce (in senso cellulare).

Anche se nessuno a quel tempo se ne rese conto, ci si trovava di fronte a un possibile mezzo di comunicazione per un calcolatore Vita a due dimensioni: alianti invece di impulsi èlettronici!

Il passo successivo si ebbe nel 1970 con la scoperta di un cannone ad alianti da parte di R. William Gosper, Jr., e di diversi suoi colleghi *(si veda l'illustrazione di questa pagina)*. Gosper, allora studente al Massachusetts Institute of Technology, mirava a vincere i 50 dollari messi in palio da Conway, attraverso la rubrica *Giochi matematici* curata da Martin Gardner in «Le Scienze», e destinati a chi per primo dimostrasse in modo definitivo che una configurazione iniziale poteva crescere illimitatamente. Il cannone ad alianti di Gosper vomitava un nuovo aliante ogni 30 mosse. Il cannone e gli alianti costituivano una popolazione sempre crescente di cellule vive.

Oltre a Gosper c'erano altri studenti al MIT che si dedicavano a Vita nel loro tempo libero. Uno di questi era Michael D. Beeler, particolarmente interessato all'analogia tra Vita e la fisica delle particelle. Beeler indirizzava un fascio di uno o più alianti contro vari bersagli, prendendo nota con cura dei risultati, a volte modesti, a volte spettacolari, delle collisioni. Faceva anche collidere tra loro, in vario modo, due fasci di alianti. La sua perseveranza è stata ricompensata da alcune osservazioni utili.

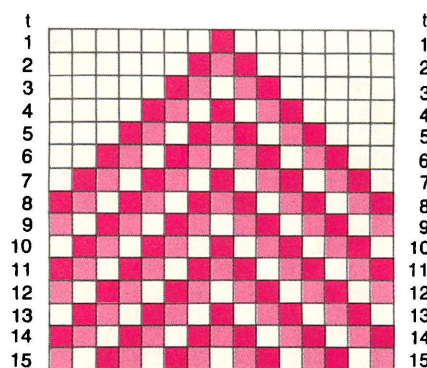
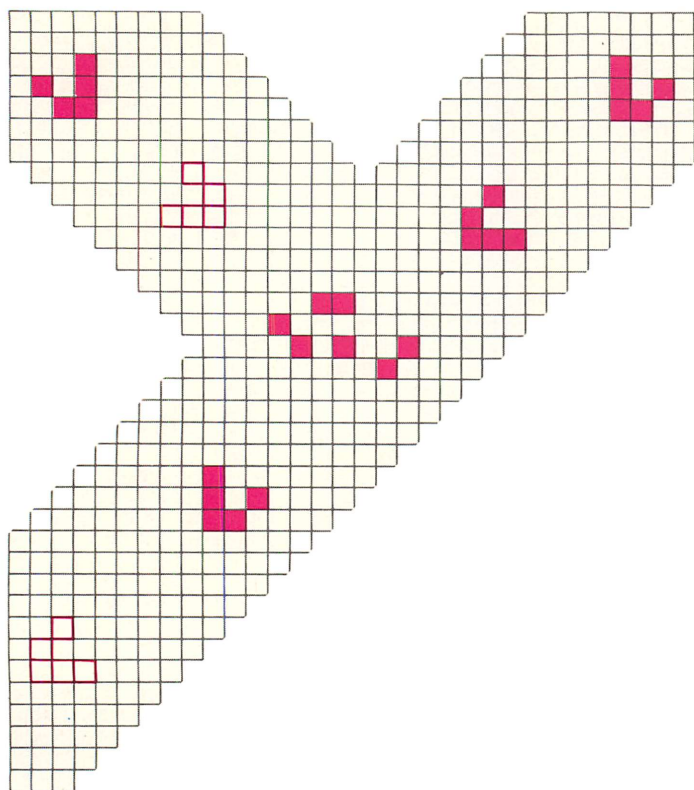
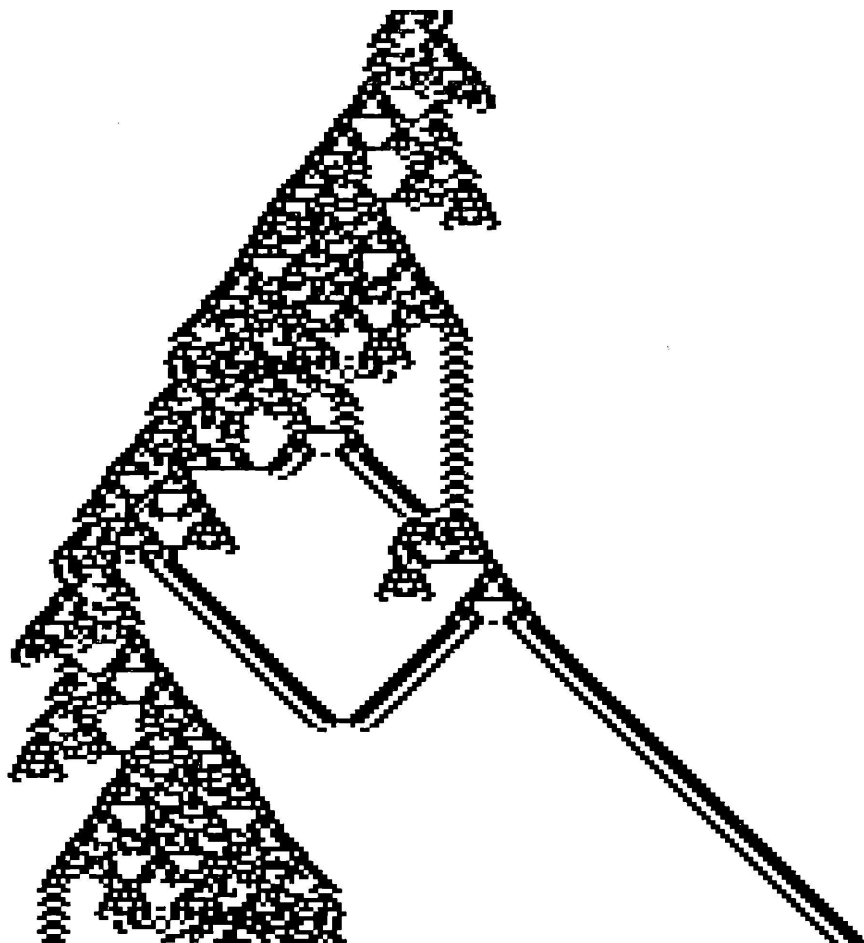
Una di queste osservazioni fu che due alianti potevano annichilarsi collidendo e questo rese possibile la costruzione di un altro componente di un calcolatore, le porte logiche. La più semplice era una porta NOT, che cambia un segnale logico in un altro: uno 0 in ingresso diviene un 1 in uscita e viceversa. La porta NOT di Vita viene costruita nel modo seguente. Si dispone il cannone in modo che invii alianti in una data direzione. I numeri binari da inviare in ingresso alla porta NOT vengono codificati in un secondo flusso di alianti indirizzato perpendicolarmente rispetto al primo. Nel flusso di ingresso un aliante può essere presente (un 1) o assente (uno 0) e questo flusso interseca quello che esce dal cannone ad alianti in modo che quando due alianti entrano in collisione tra loro si annichilino a vicenda. Questo significa che un aliante del flusso di ingresso produce un vuoto nel flusso che esce dal cannone, trasformando un aliante (1) in un non-aliante (0). L'assenza di un aliante nel flusso di ingresso consente a un aliante che esce dal cannone di passare indenne: in questo modo uno 0 viene trasformato in un 1. È interessante notare che la porta NOT non ha alcuna struttura: a parte il cannone ad alianti, che rimane fisso, la porta non è che un luogo in cui si incontrano gli alianti *(si veda l'illustrazione in alto a pagina 92)*. Anche la costruzione di altri tipi di porta, come una porta AND e una porta OR, comporta

Il cannone ad alianti di R. William Gosper in azione

flussi di alianti che interagiscono tra loro, ma è troppo complessa per poterla presentare in questa sede.

La memoria e altri registri vengono costruiti utilizzando l'interazione tra alianti e configurazioni di quattro cellule chiamate blocchi. Un singolo bit di memoria è codificato dalla posizione di un blocco. Il blocco è mosso indietro o avanti da squadre di alianti. Due alianti, su rotte scelte in modo adeguato, sono sufficienti a spostare di tre spazi in una certa direzione il blocco contro cui si scontrano. Dieci alianti, indirizzati con altrettanta cura, lo riporteranno nella posizione iniziale.

Il resto della costruzione, molto ingegnosa e interessante, si può trovare nel delizioso libro *Winning Ways for Your Mathematical Plays*, di Elwyn R. Berlekamp, John H. Conway e Richard K. Guy. Il libro è diviso in tre sezioni: giochi a due giocatori, giochi a un giocatore e giochi senza giocatori. Vita si trova nell'ultima sezione.

Scendendo il gradino finale verso la singola dimensione, vi sono da considerare solo gli spazi cellulari; è difficile immaginare come gli Ardeani potessero ridurre il loro calcolatore a un'unica linea continua. A prima vista, lo spazio cellulare sembra restrittivo quasi quanto quella linea. La diminuzione delle dimensioni, però, è compensata dal fatto che non si è più limitati a un unico insieme di regole: ci è data la possibilità di creare le nostre proprie regole. Altre compensazioni vengono dall'estrema semplicità di questi spazi lineari e dal fatto che si possono vedere alla prima occhiata centinaia di generazioni: si ponga una configurazione iniziale e si calcolino generazioni successive su linee successive scendendo lungo la pagina o lo schermo. Si svilupperà un diagramma spazio-tempo.

Un automa cellulare a una dimensione (che da qui in avanti chiameremo automa lineare) è formato da una striscia infinita di cellule che cambiano stato secondo un dato insieme di regole. Come nel gioco Vita, un orologio cosmico continua a scandire il tempo e a ogni battito ciascuna cellula assume uno stato determinato dal suo stato precedente e dallo stato precedente di cellule vicine. Per specificare un automa lineare si debbono dare due numeri, detti *k* e *r*, e un insieme di regole per derivare lo stato successivo di una cellula. Il primo numero, *k*, stabilisce quanti stati possa assumere ciascuna cellula. In Vita ci sono appena due stati e quindi *k* è uguale a 2; tra gli automi lineari da considerare sono comuni valori di *k* maggiori. Il secondo numero, *r*, si riferisce al raggio dell'intorno usato per calcolare lo stato successivo di una cellula. Lo stato attuale di una cellula e lo stato dei suoi *r* vicini da entrambi i lati determinano lo stato successivo della cellula. Per esempio, se *r* è uguale a 2 e *k* è uguale a 3, una

Un cannone ad alianti nell'automa lineare Onda

Una porta NOT nel gioco Vita: quando alianti sono presenti in ingresso, sono assenti in uscita

Un primo piano del cannone ad alianti unidimensionale di James K. Park

regola potrebbe dire che quando l'intorno di una cellula è il seguente

|0|2|1|1|0|

lo stato successivo occupato dalla cellula centrale è

|2|

L'insieme di regole che definisce un dato automa lineare deve decidere il destino di una cellula per ogni possibile configurazione del suo intorno. A seconda della dimensione di r e di k, il numero di possibili insiemi di regole può diventare enorme. Per esempio, con i valori modesti di k e di r detti sopra, ci sono più automi lineari che non atomi nell'universo conosciuto.

Chiaramente, ogni persona di questo pianeta può scegliersi il proprio automa lineare personale. Io me ne sono già scelto uno che, per ragioni che risulteranno subito chiare, si chiama Onda. In Onda, ogni cellula può assumere tre stati; ogni intorno è formato da tre cellule, una centrale e due che la affiancano. Le regole di Onda sono ragionevolmente chiare e facilmente programmabili:

1. Se una cellula è nello stato 0, il suo stato successivo sarà 2 se la somma degli stati delle cellule che le stanno a fianco è uguale o superiore a 2. Altrimenti il suo stato successivo sarà 0.
2. Se una cellula è nello stato 1, il suo stato successivo sarà 0.
3. Se una cellula è nello stato 2, il suo stato successivo sarà 1 se una o l'altra delle cellule che le stanno a fianco è nello stato 0. Altrimenti il suo stato successivo sarà 2.

Sono certo che Onda non prenderà il posto di Vita; è stato inventato solo per illustrare alcune fra le possibilità più interessanti offerte dagli automi lineari. Per esempio, Onda ha alianti semplici e un cannone ad alianti ancora più semplice (*si veda l'illustrazione in basso nella pagina precedente*).

Supponiamo che lo spazio cellulare di Onda sia totalmente nello stato 0, fatta eccezione per due cellule adiacenti: quella di sinistra è nello stato 2 e l'altra è nello stato 1. Alla generazione successiva, questa configurazione si sarà spostata di una cellula verso sinistra. Indisturbato, l'aliante di due cellule continuerà per sempre a propagarsi silenziosamente verso sinistra. Se si scambiano tra loro gli stati delle due cellule, si crea un aliante che si muove verso destra. Il cannone ad alianti è formato da una singola cellula nello stato 2; quella cellula passa attraverso gli stati 1, 0 e poi di nuovo 2, generando una coppia di alianti a ogni ciclo. Mi chiedo se qualcuno sarà capace di trovare in Onda un cannone che emetta alianti in un'unica direzione.

L'attivazione simultanea di una coppia di questi cannoni ad alianti produce

strani effetti. Nel processo di mutua annichilazione, le esplosioni spediscono alianti in entrambe le direzioni. Se i cannoni distano tra loro un numero pari di cellule, quegli alianti si allontanano in entrambe le direzioni. Altrimenti, rimane al centro un unico cannone che spara alianti in un flusso interminabile.

Onda è solo un automa lineare. E gli altri? Il numero di possibili insiemi di regole da considerare si può ampiamente ridurre adottando quelli che Wolfram chiama totalistici. Qui lo stato successivo di una cellula è determinato solo dalla somma degli stati dell'intorno della cellula, incluso lo stato della cellula data. Il numero di possibili somme varia da 0 a m, dove m è il massimo valore di stato moltiplicato per la dimensione dell'intorno. Se si specifica il modo in cui queste somme determinano lo stato successivo della cellula centrale, si ha anche una completa specificazione dell'automa lineare.

C'è, per esempio, un interessantissimo automa lineare governato dalle regole totalistiche di questa tabella:

somma	5	4	3	2	1	0
stato successivo	0	1	0	1	0	0

In questo caso, k e r sono entrambi uguali a 2; i possibili valori della somma degli stati in un intorno di cinque cellule variano da 0 a 5. Wolfram chiama «numero 20» questo insieme di regole perché i sei valori dello stato successivo della tabella rappresentano il numero 20 in notazione binaria.

Vi sono in tutto 64 modi per riempire la seconda riga della tabella; ciascuno dà luogo a un automa cellulare e Wolfram li ha esaminati tutti e 64. Inutile dire che per una ricerca di questo genere è necessario un calcolatore. Per stabilire il comportamento di un dato automa lineare, Wolfram prepara una configurazione iniziale di un centinaio di cellule in stati casuali e poi sguinzaglia l'automa sulla configurazione. Per annullare gli effetti delle ampie matrici di zeri su ciascun lato della configurazione, Wolfram rende contigua l'estremità sinistra della configurazione con l'estremità destra. Questo trasforma la configurazione in un cerchio e la sua storia diviene un cilindro, ma il risultato non cambia rispetto a quello che si avrebbe se la configurazione casuale iniziale venisse ripetuta indefinitamente in entrambe le direzioni dello spazio cellulare originario. In ogni caso, l'effetto degli automi lineari su queste configurazioni d'ingresso casuali è sorprendentemente uniforme. Ogni automa ricadrà in una delle quattro ampie classi costruite da Wolfram:

Classe 1. Dopo un numero finito di generazioni, la configurazione si riduce a un unico stato omogeneo ripetuto senza fine.

Classe 2. La configurazione si evolve in un gran numero di sottoconfigurazioni invarianti o periodiche.

Classe 3. La configurazione non sviluppa alcuna struttura. I diagrammi spazio-tempo sembrano caotici.

Classe 4. La configurazione sviluppa sottoconfigurazioni complesse localizzate, alcune di lunga durata.

Dei 64 automi in cui k e r sono entrambi uguali a 2, il 25 per cento appartiene alla classe 1, il 16 per cento alla classe 2, il 53 per cento alla classe 3 e solo il 6 per cento alla classe 4. Wolfram sospetta che si potrebbero trovare dei calcolatori nella classe 4. Per esempio, l'automa lineare con numero di codice 20 appartiene alla classe 4.

Quasi per incoraggiare Wolfram nella ricerca, l'automa di codice 20 ha gentilmente rivelato la presenza di alianti: 10111011 e 1001111011. Entrambi gli alianti si muovono verso destra. Le regole totalistiche sono sempre simmetriche; per ottenere alianti che si muovono verso sinistra basta rovesciare queste configurazioni. C'è un cannone per questi alianti nello spazio di codice 20? Wolfram ritiene di sì. Ma c'è un altro spazio in cui non è ancora riuscito a trovare nemmeno un aliante! È lo spazio di codice 792. Se scriviamo questo numero in notazione ternaria, otteniamo un insieme di regole per un automa lineare a 3 stati:

somma	6	5	4	3	2	1	0
stato successivo	1	0	0	2	1	0	0

La ricerca di alianti e di cannoni ad alianti richiede la lettura delle note scritte da Wolfram su mio invito e riportate a pagina 151.

La ricerca di alianti e di cannoni ad alianti si incentra su un gran numero di automi lineari che si ritengono universali dal punto di vista del calcolo. In altre parole, si tratta di automi lineari in grado di agire come un calcolatore. Oltre agli automi di codice 20 e di codice 792, c'è l'automa lineare a due stati (r = 3, numero di codice 88) in cui è già stato trovato un cannone ad alianti.

La scoperta del cannone in quell'automa è dovuta a James K. Park, studente alla Princeton University. I lettori che volessero vedere in azione il cannone ad alianti di Park devono scrivere un semplice programma per mostrare le generazioni di un automa lineare e per derivare una generazione dalla successiva. È facile realizzare con un programma di questo genere l'automa lineare 88: quando siete pronti, immettete la configurazione iniziale 1111111111011 e osservatela espandersi e contrarsi. Il cannone emette un aliante in ciascuna direzione ogni 238 generazioni.

Invece di realizzare un particolare automa, si consiglia ai lettori di scrivere un programma che accetti come ingresso regole totalistiche arbitrarie. Questo è facile per valori prefissati di k e r, e quasi altrettanto facile se si ammettono variazioni per questi parametri. Usate una speciale matrice chiamata *tabella* per le regole e due matrici lineari chiamate *nuovecellule* e *vecchiecellule* per conservare le nuove e le vecchie configurazioni attualmente elaborate. Le due matrici possono essere costruite dell'ampiezza voluta, ma bisogna ricordare che uno schermo (a seconda del tipo) può mostrare solo una parte limitata di queste matrici. Visualizzate quanto più possibile della parte centrale di *nuovecellule* e seguite ciclicamente i seguenti passi: trasferite il contenuto di *nuovecellule* in *vecchiecellule;* controllate *vecchiecellule;* per il suo i-esimo membro sommate il valore di *vecchiecellule* da $i - r$ a $i + r$. Cercate la somma risultante nella *tabella* e trasferite il valore di stato così trovato nell'i-esimo membro di *nuovecellule.* Riportate poi il ciclo computazionale nella fase di visualizzazione e ripetetelo. Le visualizzazioni possono essere successive o stazionarie. Nel primo caso risulta un diagramma spazio-tempo; nel secondo si vede una specie di film a una dimensione.

Come si è osservato in precedenza, un automa lineare è detto universale dal punto di vista del calcolo se, lungo il succedersi delle generazioni, c'è una

Il cannone di Park che emette alianti a destra e a sinistra

configurazione iniziale in grado di agire come un calcolatore. Parte della configurazione iniziale è l'ingresso e parte di qualche configurazione successiva è l'uscita. È davvero possibile costruire un calcolatore di questo genere partendo da cannoni ad alianti e da vari altri pezzi di hardware cellulare? Wolfram ritiene di sì. In uno degli automi lineari attualmente sotto studio si sono trovati alianti che passano attraverso determinate sottoconfigurazioni stabili. Questo fa sperare che la trasmissione di informazione all'interno di un calcolatore lineare non debba necessariamente essere bloccata da componenti estranei a quell'informazione. Oltre a Wolfram, altri ricercatori stanno compiendo esperimenti con automi lineari. Tra di essi c'è Kenneth Steiglitz, di Princeton, il quale ha trovato alianti con proprietà simili a quelle dei solitoni. Questi alianti possono addirittura sorpassarsi l'un l'altro!

Alla fine troveremo, in un regno cellulare a una dimensione, strutture persistenti capaci di muoversi, di immagazzinare e di manipolare informazione? Forse no. Può darsi che ci sia qualcosa di molto differente nella natura stessa del calcolo cellulare nella singola dimensione, qualcosa che ci richieda di considerare il calcolo in modo interamente nuovo.

Wolfram si è sbizzarrito a immaginare come gli automi lineari possano gettar luce sul comportamento degli automi in generale, e come questi possano a loro volta portare a una migliore comprensione dei processi naturali. Supponiamo di voler trovare un automa lineare che imiti qualche processo naturale, per esempio un flusso turbolento di liquido o di gas, il moto di particelle sotto l'influenza di forze, o addirittura un processo biologico come la crescita. Supponiamo inoltre che l'automa si riveli universale dal punto di vista del calcolo. In altre parole, non solo il suo spazio può contenere una struttura esplicita che calcoli, ma lo stesso spazio contiene anche implicitamente il calcolatore: qualsiasi tentativo di prevedere il comportamento dell'automa sarebbe per definizione equivalente al tentativo di prevedere le azioni di un calcolatore di uso generale. Questo in generale non può essere fatto se non dallo stesso tipo di congegno, un altro calcolatore di uso generale. Ne consegue che non sarebbe possibile alcuna scorciatoia per prevedere il comportamento del corrispondente sistema naturale. I suoi processi sarebbero irriducibili dal punto di vista del calcolo, nel senso che il meccanismo di previsione (sia esso una formula o un calcolatore) deve simulare più o meno direttamente il sistema in questione.

Questa conclusione ci riporta a Onda. Gli alianti di una popolazione iniziale vanno in direzioni casuali e rimbalzano uno contro l'altro. A volte le loro collisioni danno luogo a un cannone ad alianti, che produce altri alianti, e a volte le loro collisioni non producono nulla. Questo mi fa pensare a un universo unidimensionale in miniatura pieno di particelle che ondeggiano avanti e indietro. Ho rovesciato allora l'ordine della ricerca di Wolfram: partendo da un automa a striscia che si comporta vagamente come un sistema di particelle (per quanto strane), il mio sogno è che Onda sia universale dal punto di vista del calcolo. Visto che si tratta del mio automa personale, forse questo sogno è solo mio.

III. Geometria e grafica

Tutti sicuramente siamo rimasti colpiti dai raffinati effetti realizzati al calcolatore, nelle pellicole della Lucasfilm, o in *Tron* della Walt Disney: l'elaborazione di immagini è una delle aree più sviluppate all'interno dell'informatica tradizionale e ha già dato contributi notevoli, al di fuori dell'ambito spettacolare, nell'informatica medica, nell'elaborazione delle immagini da satellite, nelle applicazioni per l'ingegneria e l'architettura, nel restauro.

Una buona grafica è essenziale per la riuscita di quasi tutti i tipi di giochi, e questa osservazione basterebbe da sola a giustificare la presenza di questa parte (quasi tutti i calcolatori domestici e personali, ormai, hanno capacità grafiche avanzate, che solo di rado però vengono sfruttate appieno), ma negli articoli che seguono c'è molto di più. Un trattamento intelligente delle immagini si è rivelato uno degli scogli più difficili da superare: il riconoscimento di un volto, per esempio, operazione che a noi risulta così familiare che possiamo compierla quasi sempre senza doverci riflettere, è invece un compito di elevata difficoltà per una macchina. Il problema del riconoscimento di forme attende ancora soluzioni di maggiore respiro, dalle quali dipenderà in futuro lo sviluppo della robotica. Anche in quest'ottica si può intendere, per esempio, il lavoro sulle caricature al calcolatore.

Ci sono poi tutti gli aspetti matematici (geometrici in particolare): l'esplorazione dei frattali, iniziata con i lavori di Benoit Mandelbrot, per esempio, riserva sorprese a ogni passo, per non parlare del piacere estetico che procurano costruzioni basate su questa tecnica matematica. È caratteristico che le idee della matematica estendano spesso il loro campo d'azione molto al di là della loro applicazione originale o principale: per quanto discussa, la teoria dei frattali di Mandelbrot estende le sue ramificazioni un po' in tutti i settori della fisica.

Il primo degli articoli di questa parte ci riporta, invece, al tema dei rapporti fra gioco e istruzione, in una delle forme più collaudate (sia pure solo a livello elementare): la geometria della tartaruga e il linguaggio di programmazione LOGO, realizzato da Seymour Papert al MIT e ispirato alle teorie psicologiche di Jean Piaget. «Cinque secoli fa - scrivono H. Abelson e A. Di Sessa nella prefazione al loro *La geometria della tartaruga* - l'invenzione della stampa provocò un radicale rinnovamento nella natura dell'istruzione. Portando le parole del maestro a coloro che non potevano udirne la voce, la tecnologia della stampa distrusse la concezione per cui l'istruzione doveva essere riservata a coloro che potevano permettersi di pagare insegnanti privati. Oggi ci avviamo verso una nuova rivoluzione tecnologica il cui impatto sull'istruzione può avere la stessa portata dell'invenzione della stampa: la comparsa di potenti computer sufficientemente economici da poter essere usati dagli studenti per imparare, giocare e condurre ricerche. È nostra speranza che questi potenti ma semplici strumenti per la creazione e l'esplorazione di ambienti altamente interattivi distruggano le barriere che si frappongono alla produzione della conoscenza così come la stampa distrusse le barriere che si frapponevano alla sua trasmissione.

«Questa speranza è qualche cosa di più del desiderio di far provare agli studenti la gioia della scoperta e dell'interscambio tra ricercatore e ricerca che caratterizza l'indagine scientifica. Come Piaget, Dewey e la Montessori, siamo convinti che il coinvolgimento e la partecipazione dell'individuo siano essenziali perché l'apprendimento sia realmente effettivo. Quasi tutti i programmi di matematica, tuttavia, sono dedicati alla pratica meccanica degli algoritmi e alla riproposizione di vecchi teoremi. È raro che gli studenti abbiano la possibilità di avvicinarsi alla matematica "facendola" anziché studiandola soltanto; è raro che possano indagare su nuovi fenomeni, formulare ipotesi originali e dimostrare teoremi originali. Il calcolo automatico - e in particolar modo l'attività di programmazione - è in grado di offrire agli studenti molte buone opportunità per "fare" matematica senza prima dover essere padroni di un grande apparato di nozioni. E ciò che ci proponiamo di dimostrare è come un approccio di tipo computazionale possa mutare il rapporto tra studenti e conoscenze matematiche.

«L'esperienza è un ingrediente importante della scoperta. La quantità di fenomeni che gli studenti possono esaminare per conto proprio con l'aiuto dei modelli costruiti al calcolatore dimostra come i computer possano aprire una fase dell'istruzione in cui "apprendimento attraverso la scoperta" diventa qualche cosa di più di una frase carica di buone intenzioni. I computer possono introdurre nell'apprendimento l'elemento essenziale della sorpresa; infatti, malgrado la diffusa convinzione che un computer non possa mai sorprendere il proprio programmatore poiché fa solo ciò che è stato programmato a fare, anche algoritmi semplicissimi possono produrre e spesso producono risultati inaspettati e sorprendenti. Ottenere uno di questi risultati, studiarlo e capire da che cosa è prodotto può rappresentare un'avventura a finale aperto molto diversa della maggior parte dei "metodi di scoperta" normalmente usati nell'insegnamento, dove l'insegnante sa già in anticipo che cosa esattamente si deve "scoprire"».

Guardando la geometria dal di dentro, a passeggio con una tartaruga

di Brian Hayes

Le Scienze, aprile 1984

Che cos'è una circonferenza? È il caso limite di un poligono di *n* lati, quando *n* cresce fino all'infinito? È il caso speciale di un'ellisse in cui i due fuochi coincidono? È il luogo dei punti del piano equidistanti da un punto dato?

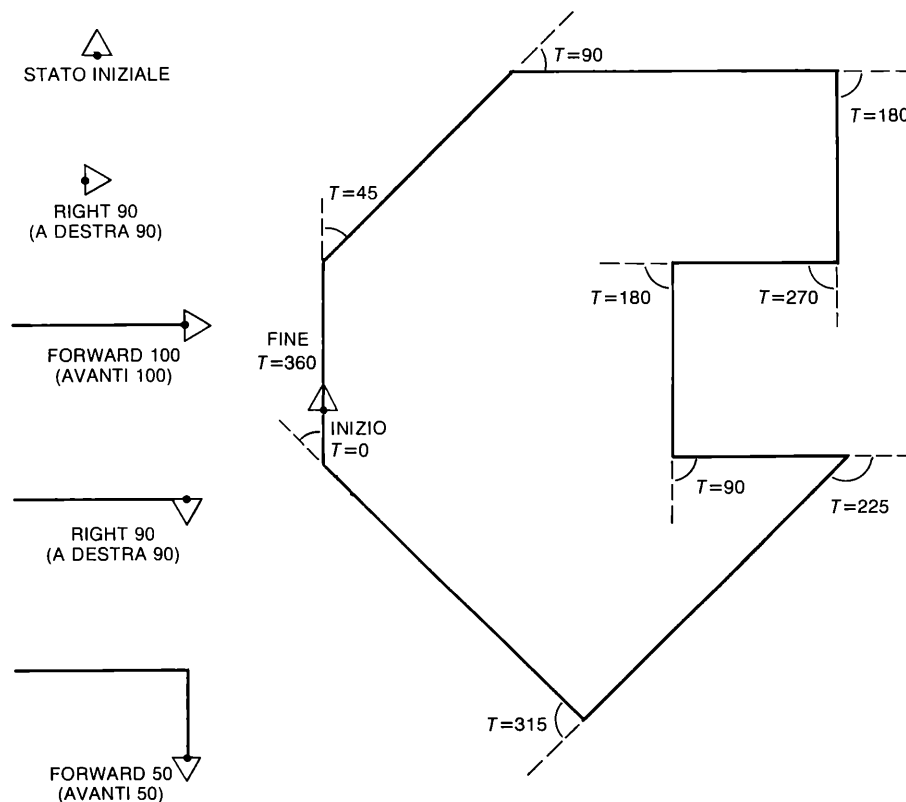
Tutte queste definizioni (e se ne potrebbero dare delle altre) sono, ovviamente, corrette. Si consideri questa definizione: una circonferenza è la figura che si ottiene avanzando per un po', poi piegando per un po' a destra e ripetendo questa successione di passi finché non si è coperto esattamente un angolo di 360 gradi. Quest'ultima definizione è fondamentalmente diversa dalle altre: invece di descrivere una circonferenza o di specificare alcune delle sue proprietà, dà una procedura per costruirne una. Inoltre la stessa procedura ha una caratteristica speciale: è formulata interamente in termini di proprietà «locali» della circonferenza. La curva si può creare semplicemente spostandosi nelle immediate vicinanze; non c'è bisogno di una «visione» complessiva. Non c'è bisogno di sapere dove si trovi il centro della circonferenza e neppure di determinare il raggio.

Procedure di questo tipo costituiscono l'essenza di un nuovo modo di considerare la geometria. È stata definita geometria esperienziale, perché invita a immaginare di muoversi in un mondo di figure geometriche, mentre le altre impostazioni della geometria tendono a collocare le figure in uno spazio separato dall'osservatore. La differenza è pressappoco quella che esiste tra esplorare un paesaggio e leggere una mappa. L'impostazione esperienziale è particolarmente adatta per esplorare idee geometriche con l'aiuto di un calcolatore. Le definizioni procedurali possono essere trasformate facilmente in programmi per calcolatore che si possono eseguire per disegnare le forme volute.

Il nuovo modo di considerare la geometria è chiamato anche «*turtle geometry*» (geometria della tartaruga). È strettamente collegato col linguaggio di programmazione Logo che, a sua volta, trova la sua origine nel Massachusetts Institute of Technology. Il Logo è stato ideato negli anni sessanta da Seymour Papert del MIT come linguaggio che aveva come scopo principale quello di avvicinare i bambini al calcolatore. Da allora molti altri hanno contribuito a svilupparlo e ad applicarlo sia in campo educativo che in altri campi. Tra questi Harold Abelson e Andrea DiSessa del MIT, che hanno diffuso le idee che stanno alla base della geometria della tartaruga in un'opera divulgativa degna di nota: *Turtle Geometry: The Computer as a Medium for Exploring Mathematics.*

La tartaruga era originariamente un congegno meccanico, un piccolo veicolo a ruote i cui movimenti potevano essere controllati attraverso istruzioni battute sulla tastiera di un calcolatore. La prima creatura cosiffatta fu costruita dal neurofisiologo americano W. Grey Walter alla fine degli anni quaranta. Aveva un coperchio ricurvo che assomigliava al guscio di una tartaruga. Una tartaruga meccanica si può muovere avanti e indietro e può cambiare direzione girandosi sul posto. Sul telaio può essere montata una penna che lascia una traccia del percorso della tartaruga quando viene fatta muovere su di un foglio di carta. Oggi queste «tartarughe di pavimento» sono molto meno comuni delle «tartarughe di schermo» che si muovono e disegnano sulla superficie di un tubo catodico. Sullo schermo la tartaruga è rappresentata da una semplice forma triangolare, che si muove in risposta a comandi o programmi inseriti attraverso la tastiera.

La maggior parte dei metodi per disegnare con il calcolatore fanno uso di un sistema globale di coordinate, generalmente cartesiano, con due assi perpendicolari. Ogni punto dello schermo ha una posizione definita rispetto a un'origine, per esempio l'angolo in basso a sinistra, in cui gli immaginari assi si incontrano. Anche le direzioni nel piano sono assolute. Si può dare il comando di disegnare una retta specificando i due estremi; per esempio, dando i punti {0,0} e {100,0} si potrebbe disegnare una retta orizzontale in fondo allo schermo, lunga 100 unità.

Si dovrebbe sottolineare che, in qualsiasi sistema di coordinate globali, l'effetto di un comando non dipende dalla sequenza dei comandi precedenti. La situazione è del tutto diversa in un sistema di geometria della tartaruga. Si può disegnare una retta lunga 100 unità dando alla tartaruga un'istruzione del tipo FORWARD 100 (*forward* significa «avanti»). In quale punto dello schermo compaia la retta e quale sia il suo orientamento dipende completamente da dove si trovava la tartaruga al momento in cui il comando è stato dato. In ogni momento la tartaruga ha una posizione e una orientazione. Il comando FORWARD 100 la fa avanzare

***Alcuni comandi della tartaruga* (a sinistra) *e il concetto di rotazione totale* (a destra)**

di 100 unità rispetto alla sua posizione attuale nella direzione definita dalla sua orientazione (cioè nella direzione verso cui è «girata»). Lo stesso comando, quindi, ha un effetto del tutto diverso quando la tartaruga è in un diverso stato iniziale.

Si può creare un semplice sistema di geometria della tartaruga con due soli comandi: FORWARD e RIGHT «destra». FORWARD è seguito da un numero che specifica quanti «passi» deve fare la tartaruga. Anche RIGHT è seguito da un valore che indica di quanti gradi la tartaruga deve deviare a destra rispetto alla sua orientazione attuale. Da una serie di mosse avanti e a destra può risultare il disegno di una qualsiasi figura piana. (Si noti che una svolta a sinistra di 90 gradi si può specificare in due modi: come RIGHT 270 o RIGHT −90.) Ciononostante un sistema pratico comprende generalmente anche i comandi BACK o REVERSE («indietro») e LEFT («sinistra»). PENUP («penna su») e PENDOWN («penna giù») indicano se la tartaruga traccia o meno una linea durante il suo cammino. Sono compresi anche alcuni altri comandi che informano sullo stato della tartaruga e offrono un mezzo per specificare le coordinate assolute, ma non sono essenziali.

Il tipo di geometria che si fa con la tartaruga viene definito da un punto di vista formale geometria differenziale finita. Finita perché la tartaruga può compiere solo passi discreti; differenziale, perché tutti i movimenti sono definiti in termini di differenza tra la posizione e l'orientazione presenti e quelle successive. Il che equivale a dire che è un tipo di geometria che riguarda solo le proprietà locali di rette, curve, superfici; non si fa nessun riferimento a punti lontani o alle proprietà globali di una figura geometrica. Ne segue che la geometria della tartaruga è più utile per esplorare le proprietà «intrinseche» delle figure geometriche, quelle che possono essere definite interamente all'interno delle figure stesse.

Un'idea che potrebbe risultare difficile da formulare in un sistema di coordinate e che invece si presenta molto chiaramente nella geometria della tartaruga è quella della curvatura. In un sistema di coordinate cartesiane la curvatura di una linea si potrebbe definire come la variazione di inclinazione della linea; l'inclinazione, a sua volta, si potrebbe definire come la variazione della coordinata y come funzione della x. La tartaruga può rendersi conto della curvatura in un modo molto più semplice: è la rotazione totale per unità di distanza percorsa. Allora, in un cerchio definito dalla ripetizione delle istruzioni FORWARD 1, RIGHT 1, la curvatura è sempre 1.

Il concetto di rotazione totale porta altri interessanti risultati. Si considerino i familiari teoremi della geometria euclidea che danno la somma degli angoli interni di un poligono convesso. Per un triangolo la somma è 180 gradi, per un quadrilatero 360, per un pentagono 540 e così via. In altre parole, l'angolo interno è sempre un multiplo intero di 180 gradi e tale intero è uguale al numero di lati meno 2. La rotazione totale della tartaruga è misurata in base a un angolo diverso: non è né interno né esterno, ma è dato dal cambiamento di orientazione a ogni vertice. Si possono formulare anche teoremi sulla rotazione totale in un poligono. Sono un po' diversi da quelli sulla somma degli angoli interni e anche più generali.

Il teorema principale stabilisce che la rotazione totale, quando la tartaruga traccia un poligono, deve essere un multiplo intero di 360 gradi. La dimostrazione è semplice ed evidente. Se una figura è chiusa, come lo è ogni poligono, la tartaruga deve alla fine ritornare al suo punto iniziale e, quando lo raggiunge, deve avere esattamente la stessa orientazione che aveva all'inizio. (Questa osservazione funge, in realtà, come definizione effettiva di chiusura geometrica.) Tuttavia, se l'orientazione è la stessa, la rotazione totale deve essere 0 o 360 o qualche multiplo di 360 gradi.

Per un poligono convesso si può dimostrare un risultato più forte, precisamente che la rotazione totale è esattamente 360 gradi. (Un poligono si dice convesso, se una linea che congiunge due punti qualsiasi sui suoi lati non passa mai all'esterno di esso.) Lo stesso teorema, tuttavia, si applica a qualsiasi poligono, compresi quelli che non sono convessi (come un pentagono a stella). Si applica persino a figure chiuse che hanno lati curvi e pertanto non sono poligoni.

In una geometria delle coordinate la somma degli angoli interni ha pochi collegamenti ovvi con altre idee, mentre nella geometria della tartaruga il concetto di rotazione totale è uno strumento potente, ricco di ampie applicazioni. Per esempio, Abelson e DiSessa dimostrano che lo si può usare per analizzare la topologia di un cammino chiuso. Un cammino topologicamente semplice può avere un numero qualsiasi di spigoli, curve e circonvoluzioni, purché non incroci mai se stesso; come un poligono semplicemente convesso, ha una rotazione totale di 360 gradi. L'aggiungere un cappio (e un incrocio) al cammino porta la rotazione totale a zero o a 720 gradi, a seconda della direzione del cappio. Ogni cappio che viene aggiunto comporta l'aggiunta o la sottrazione di 360 gradi. Grazie a questa proprietà si può determinare la topologia del cammino, osservandolo dal punto di vista della tartaruga, anche se in un momento qualsiasi è visibile solo una sezione microscopica di esso.

La rotazione totale sta anche alla base di un algoritmo che permette alla tartaruga di uscire da un labirinto. Abelson e DiSessa lo definiscono come segue.

«1. Si scelga una direzione iniziale arbitraria, la si chiami "nord" e ci si rivolga verso di essa.

«2. Si avanzi in "direzione nord" finché non si incontra un ostacolo.

«3. Si devii a sinistra finché l'ostacolo rimane sulla destra.

«4. Si aggiri l'ostacolo, mantenendolo sempre a destra finché la rotazione totale (compresa quella iniziale al passo 3) non risulta zero.

«5. Si torni al passo 2.»

Con questo metodo la tartaruga può risolvere qualsiasi labirinto (ovviamente purché abbia effettivamente una via di uscita). La procedura funziona perché l'unico modo per intrappolare la tartaruga è quello di farla entrare in un cappio senza via d'uscita e registrando la rotazione totale si evita tale pericolo. Si noti che ancora una volta è stato risolto un problema generale, cioè trovare la via d'uscita dal labirinto, benché la tartaruga disponga solo di informazioni sul suo ambiente locale: non si ha accesso a vedute aeree del labirinto. La procedura universale di risoluzione del labirinto viene chiamata algoritmo di Pledge, da John Pledge di Exeter, in Inghilterra, che l'ha elaborata all'età di 12 anni.

Ulteriori applicazioni della rotazione totale si possono ricavare se si fa muovere la tartaruga su superfici non piane. Su di una sfera, per esempio, la rotazione totale per un cammino chiuso risulta minore di 360 gradi: quanto minore, poi, dipende dalla lunghezza del cammino o, per essere più precisi, dall'area che racchiude. I lettori che abbiano familiarità con le geometrie non euclidee fiorite nel XIX secolo riconosceranno ciò che si verifica in questo caso: la deviazione è una misura della curvatura della superficie. In questo caso è importante il fatto che la misura possa essere ottenuta semplicemente attraverso informazioni locali e dall'interno. La curvatura che si desidererebbe maggiormente misurare è quella dello spazio-tempo dell'universo, che ovviamente dovrebbe essere ricavata da osservazioni compiute all'interno dell'universo. Abelson e DiSessa esplorano questo processo nel capitolo finale del loro libro che dà una formulazione, nella geometria della tartaruga, della teoria generale della relatività.

Essi prendono in considerazione anche il vagabondare della tartaruga sulla superficie di un cubo, che ha la stessa topologia della sfera, ma una diversa geometria. La geometria, in effetti, è bizzarra. Supponiamo che il cubo abbia il lato di 100 unità e che la tartaruga sia inizialmente al centro di una faccia, orientata parallelamente a uno spigolo. Si ripetano le istruzioni FORWARD 100, RIGHT 90 per tre volte. Si ottiene una figura chiusa con tre lati e tre angoli uguali; è anche un poligono equilatero ai cui vertici si trovano tutti angoli retti. Si tratta di un triangolo rettangolo equilatero o di un quadrato trilatero?

Un altro oggetto della geometria cubica degno di nota è il monogono, un poligono di un lato; è un cammino chiuso che si ottiene quando la tartaruga procede senza mai deviare. Anche senza far guidare la tartaruga dal calcolatore è possibile vedere come si può formare un semplice monogono: basta tracciare un «equatore» sul cubo. Una domanda più difficile è la seguente: c'è una combinazione di posizione e orientazione iniziali sulla superficie del cubo che non si chiuda a formare

un monogono quando la linea si estende all'infinito? E se l'orientazione venisse data in numeri razionali? Troverete la risposta al sesto capitolo di *Turtle Geometry*, ma Abelson e DiSessa vi scoraggerebbero dal guardarvi. A un certo punto, infatti, pongono questo avvertimento: «PERICOLO — La sezione che segue contiene matematica "prefabbricata" (già inventata da altri). Potrebbe essere dannosa per la vostra immaginazione e dovrebbe servire solo come ultima risorsa.»

La scoperta di un principio e l'esplorazione della sua estensione o generalizzazione sono caratteristiche dello stile in cui è costruita la geometria della tartaruga. Il calcolatore rende molto più facile procedere in questo modo: è facile compiere esperimenti e si possono provare variazioni sul tema con poco sforzo. È una geometria per chi ama darsi da fare.

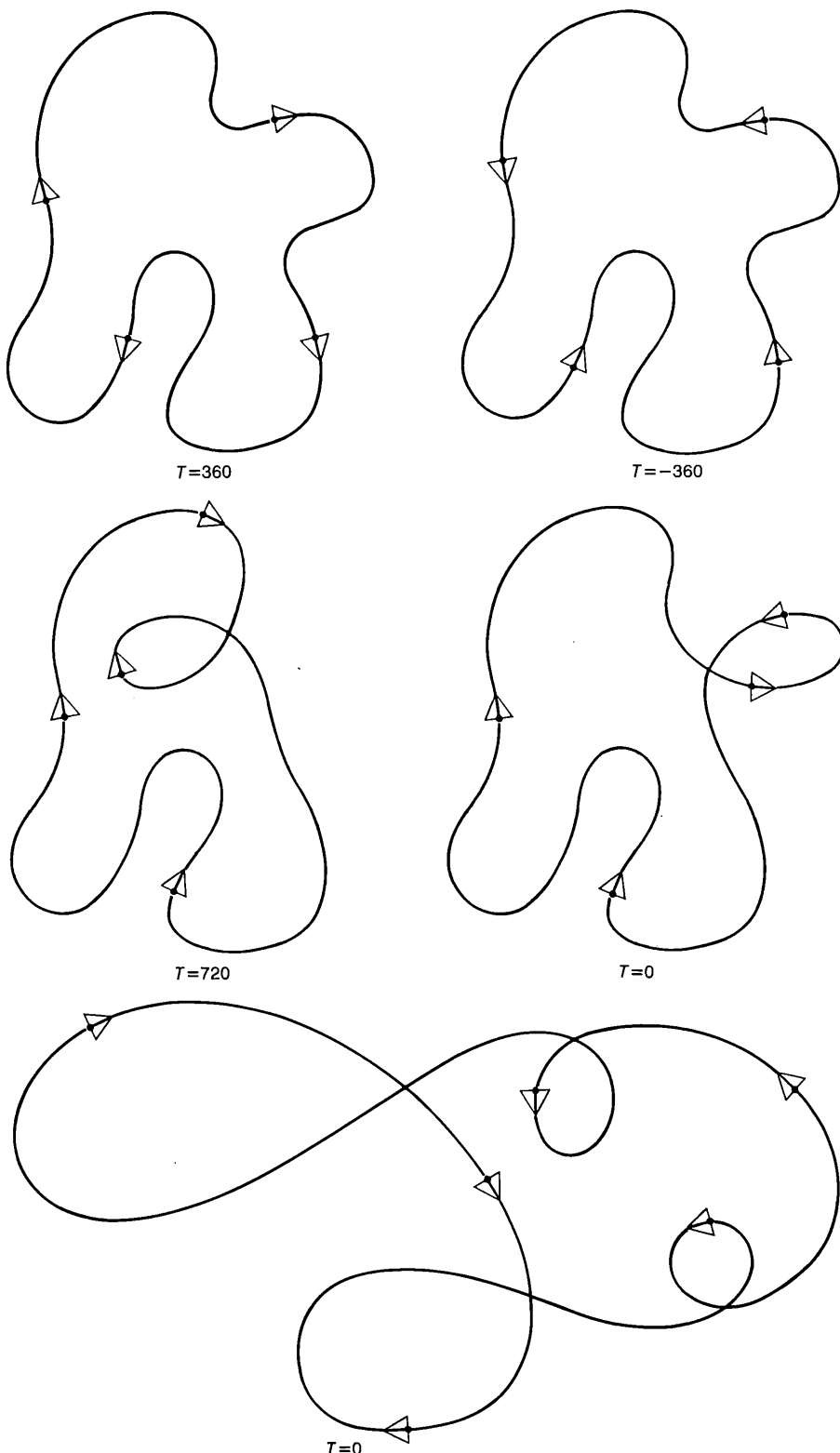
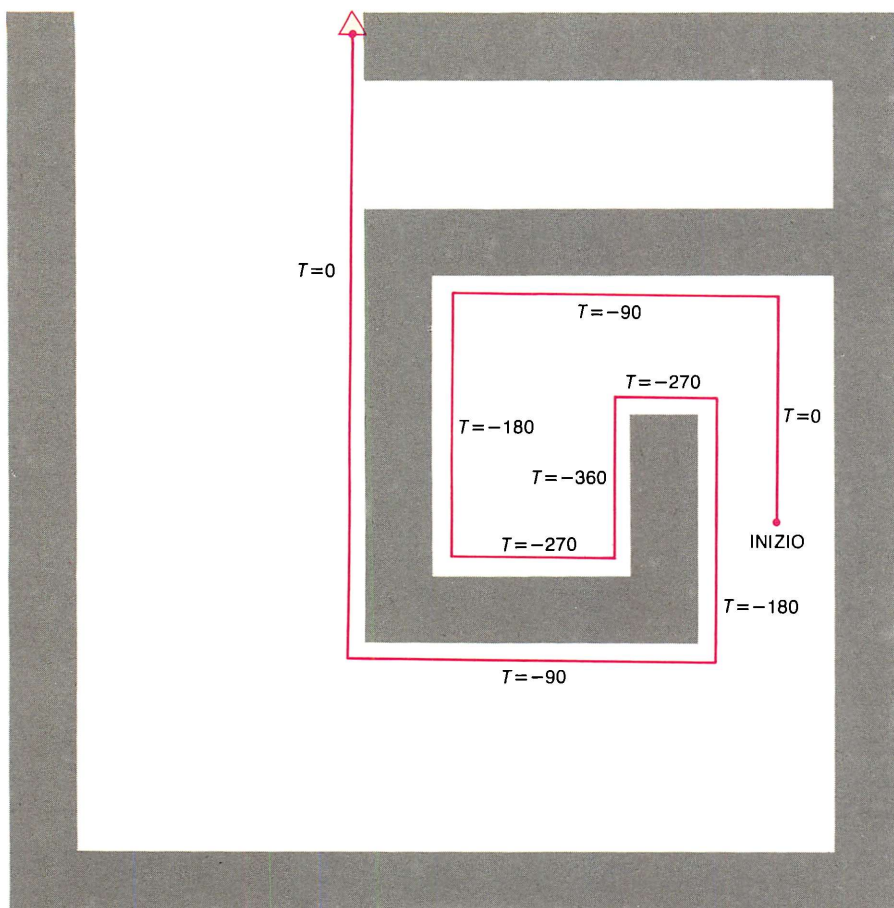
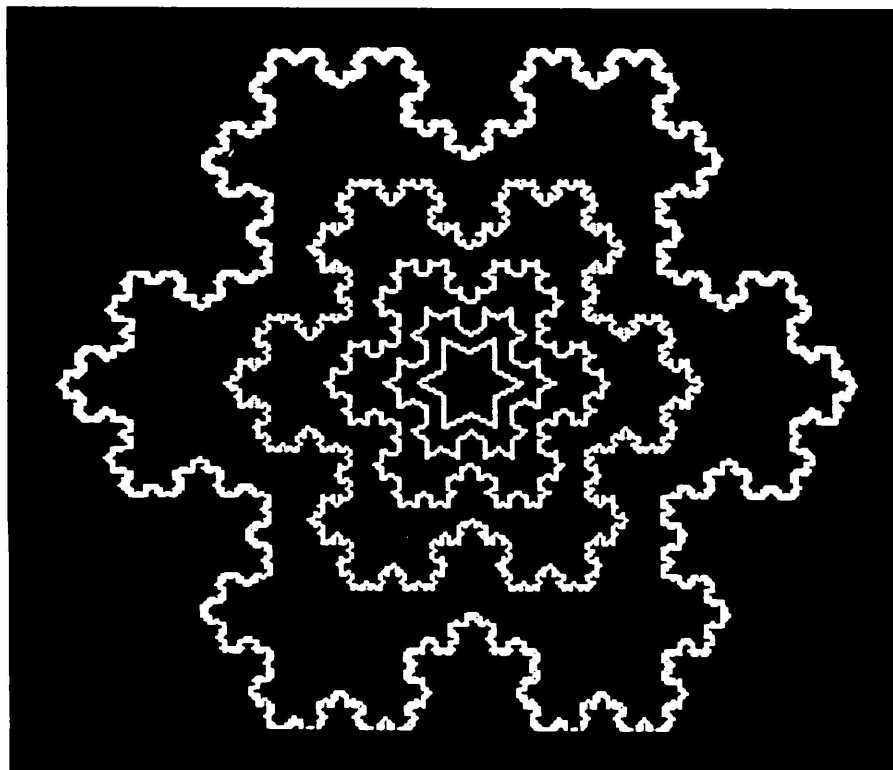
Si consideri la seguente procedura in Logo, esaminata da Abelson e DiSessa e da altri autori:

```
TO SQUIRAL :DISTANCE
FORWARD :DISTANCE
RIGHT 90
SQUIRAL :DISTANCE + 5
END
```

Qui SQUIRAL è il nome della procedura e DISTANCE è una variabile, il cui valore iniziale deve essere dato al momento dell'esecuzione della procedura. (I due punti sono una convenzione del Logo per indicare le variabili.) Alla tartaruga viene data istruzione di muoversi in avanti nella misura data da DISTANCE e di eseguire una rotazione a destra di 90 gradi; poi si richiede di ripetere la procedura SQUIRAL ma con un più alto valore di DISTANCE. Il risultato è una «spirale quadrata» che cresce verso l'esterno in direzione dei bordi dello schermo. (Nella procedura come è data qui, la spirale cresce indefinitamente anche se ne compare sullo schermo solo una parte.)

Sono possibili molte variazioni. Cambiare il valore della costante, da aggiungere a DISTANCE ogni volta che la procedura è chiamata, produce semplicemente un'alterazione della distanza tra i successivi giri della spirale. Moltiplicare per un valore costante, invece che sommarne uno, dà luogo a una spirale i cui giri si allontanano progressivamente, proporzionalmente alla loro distanza dal centro. Inserire un diverso angolo costante nell'istruzione RIGHT 90 può trasformare la spirale quadrata in una triangolare o pentagonale o esagonale. Un angolo che differisca solo di poco da 90 gradi porta a una serie di «quadrati» intrecciati che si avvolgono intorno al loro centro cosicché i loro vertici formano spirali secondarie. Un angolo molto piccolo comporta l'approssimazione a una spirale «circolare» regolare.

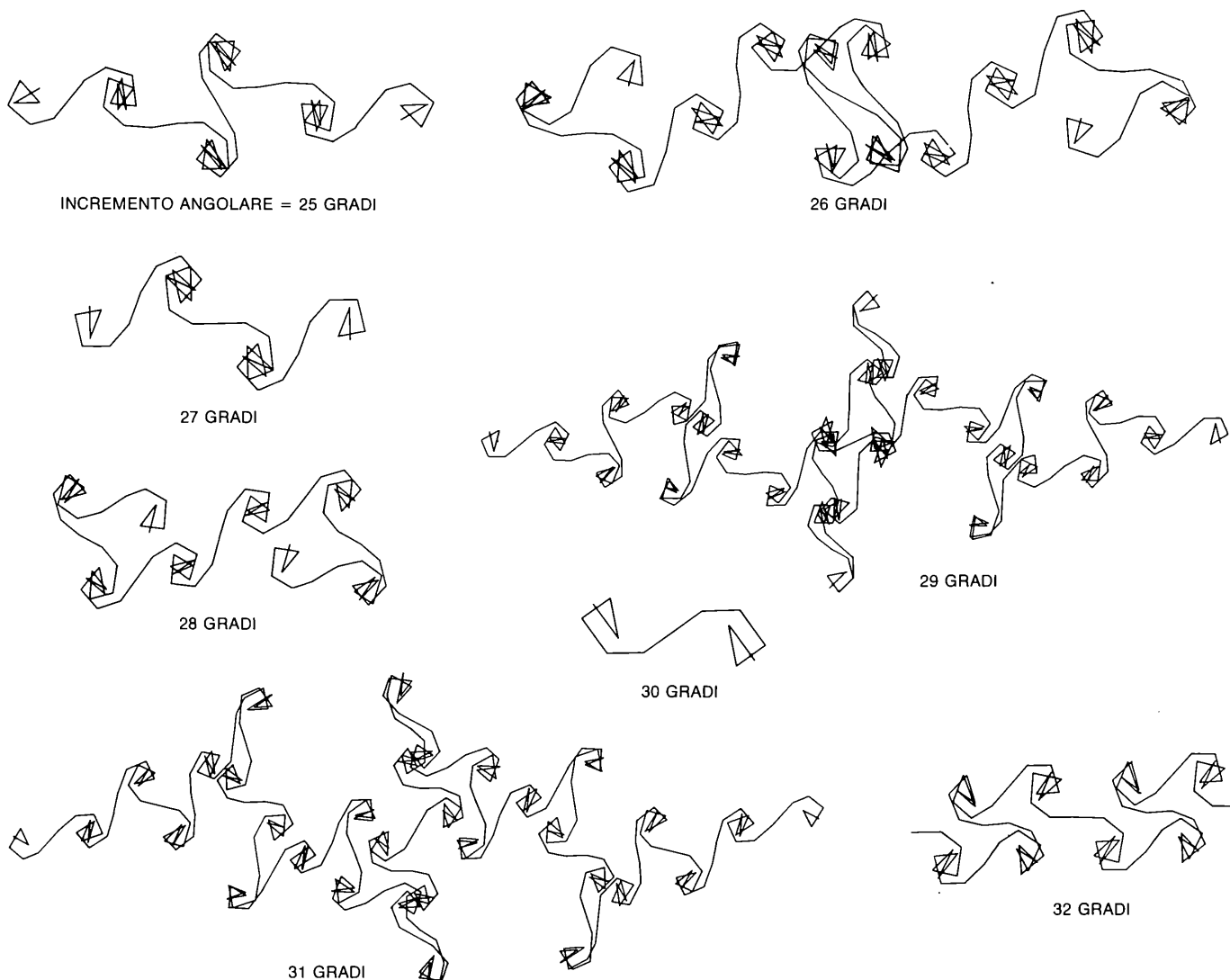
È anche possibile riscrivere la procedura in modo che ciò che cambia a ogni chiamata sia l'angolo e non la distanza. La trasformazione è notevole: invece di una sola spirale che si sviluppa continuamente verso l'esterno, la tartaruga disegna una spirale che cresce verso l'interno e, successivamente, una che cresce verso l'esterno dalla parte opposta, poi un'altra verso l'interno e così via, creando un allineamento simmetrico di spirali congiunte dai loro giri più esterni. Sotteso a tutto ciò sta il fatto che, mentre la distanza può crescere in maniera monotona, la misura angolare viene interpretata modulo 360 in modo che, aggiungendo ripetutamente una costante, si ritorna alla fine a un angolo di piccola ampiezza.

Si considerino solamente le sottoclassi di curve formate quando l'angolo iniziale è zero (*si veda la figura a pagina 101*). Tutte le configurazioni di questa classe sono simili nella loro forma base. La tartaruga crea molte spirali di direzione alternata, poi ruota, ripercorre il suo

Topologia di curve chiuse dedotta dalla rotazione totale

La tartaruga, seguendo l'algoritmo di Pledge, esce da un semplice labirinto

Una successione a incastro di curve a fiocco di neve disegnata dalla tartaruga

cammino e, dopo essere ripassata dal punto iniziale, crea un'altra schiera di spirali identica alla prima, ma ruotata di 180 gradi rispetto a essa. Dopodiché la tartaruga continua a ripercorrere lo stesso tracciato indefinitamente. Il numero di spirali della configurazione dipende solo dall'incremento angolare, ma la natura di tale relazione non è evidente. Se l'incremento angolare non è divisibile per 8, il numero delle spirali supera di 1 il più grande fra i fattori dell'incremento che non sia anche fattore di 360. Riuscite a rendervi conto del perché? Potete predeterminare la natura della configurazione, quando l'incremento è un multiplo di 8?

È il calcolatore che rende possibile un tale avvicinamento esplorativo alla famiglia delle curve a spirale. Analoghi progetti si possono intraprendere per esplorare la famiglia dei poligoni, delle tassellature del piano e delle figure ricorsive come la curva a fiocco di neve che è costituita da strutture simili ripetute a scala sempre più piccola (*si veda l'illustrazione in basso di questa pagina*). La natura procedurale della geometria della tartaruga dà un importante contributo al processo. Sarebbe per lo meno arduo scrivere un'equazione per definire l'intera struttura della curva a fiocco di neve, mentre si può rapidamente approntare una procedura per generare la curva, con alcuni passi elementari eseguiti ripetutamente.

Benché la geometria della tartaruga e il linguaggio Logo abbiano un forte legame storico, non sono affatto inseparabili. Abelson e DiSessa fanno notare che della tartaruga ci si è serviti in almeno altri due linguaggi (versioni di APL e Smalltalk) e che i programmi del loro libro sono dati non in Logo ma in un linguaggio simile che chiamano Notazione della Procedura della Tartaruga (*Turtle Procedure Notation*). Forniscono suggerimenti per creare un sistema basato sulla tartaruga in BASIC e Pascal. Alla tartaruga si adatta anche un nuovo linguaggio, chiamato Boxer, che è stato elaborato dal gruppo del Logo al MIT.

Lo stesso Logo è un linguaggio di programmazione potente e generale. Ha una grande somiglianza con il Lisp, il linguaggio per l'elaborazione di liste inventato più di 20 anni fa da John McCarthy. Le principali differenze tra il Lisp e il Logo riguardano in effetti il vocabolario e la punteggiatura. L'origine del Logo come linguaggio per bambini gli ha conferito un aspetto antropomorfico che può risultare sconcertante. Il programmatore scrive come se stesse parlando direttamente alla tartaruga e talvolta la tartaruga risponde, producendo un messaggio di errore come «I don't know how to squiral» («Non so come fare squiral»). Non ci si deve lasciar ingannare da tale semplicità e pensare che il linguaggio sia un gioco. L'antropomorfismo è voluto: è parte di una strategia per coinvolgere il programmatore, sia un bambino o un adulto, nell'esperienza della geometria della tartaruga. Quando si viene frustrati da un programma, si viene invitati a «giocare alla

Otto schiere di spirali create aggiungendo un incremento costante a un angolo iniziale nullo

tartaruga». Il legame più profondo che esiste tra la geometria della tartaruga e Logo sta nel fatto che traggono origine da una comune filosofia dell'educazione, che si basa in gran parte sul lavoro Jean Piaget e che dà moltissima importanza alle scoperte che lo studente compie autonomamente. Papert, che ha lavorato cinque anni con Piaget, dichiara le sue ambizioni in *Mindstorms: Children, Computers, and Powerful Ideas*. «Per programmare la tartaruga si comincia col riflettere su come si fa ciò che si vorrebbe venisse fatto dalla tartaruga. Pertanto, insegnare alla tartaruga ad agire o a "pensare" può condurre a riflettere sulle proprie azioni e sui propri pensieri... L'esperienza può essere dura: pensare a come si pensa trasforma il bambino in un epistemologo, esperienza questa che la maggior parte degli adulti stessi non ha fatto.»

Un microscopio al calcolatore per gettare uno sguardo sul più complesso fra gli oggetti della matematica

di A. K. Dewdney

Le Scienze, ottobre 1985

L'insieme di Mandelbrot si colloca, nella sua silenziosa complessità, al centro di una vasta distesa bidimensionale di numeri detta piano complesso. Quando si applica ripetutamente ai numeri una certa operazione, quelli all'esterno dell'insieme fuggono all'infinito, mentre quelli all'interno vanno alla deriva ondeggiando qua e là. Vicino al margine, una minuta coreografia di oscillazioni segna l'inizio dell'instabilità. È un regresso infinito di dettagli che ci colpisce per la sua varietà, la sua complessità e la sua strana bellezza.

L'insieme prende nome da Benoit B. Mandelbrot, ricercatore al Thomas J. Watson Research Center della IBM a Yorktown Heights, New York. Partendo dal suo lavoro sulle forme geometriche, Mandelbrot ha sviluppato un campo che ha chiamato geometria frattale, lo studio matematico di forme con dimensione frazionaria. In particolare, il confine dell'insieme di Mandelbrot è un frattale, ma è anche molto di più.

Con l'aiuto di un programma relativamente semplice, un calcolatore può essere trasformato in una specie di microscopio per osservare il confine dell'insieme di Mandelbrot. In linea di principio, si può effettuare uno «zoom» su qualsiasi parte dell'insieme, con qualsiasi ingrandimento (*si vedano la copertina di questo volume e le illustrazioni delle pagine 103, 104 e 105*). Da una certa distanza, l'insieme assomiglia a un tozzo otto, coperto di protuberanze e sdraiato sul fianco. L'interno della figura è sinistramente nero. Intorno c'è un alone di color bianco elettrico, che cede il posto a profondi blu e neri nelle zone esterne del piano.

Avvicinandosi all'insieme di Mandelbrot, si scopre che ogni protuberanza è una sottile figura di forma molto simile a quella della figura genitrice. Zoomando ancora di più su una di queste sottili figure, però, ci si accorge che si tratta di una configurazione del tutto diversa: una miriade di filamenti arricciati e annodati, dall'aspetto organico, si estende in file e spirali. Ingrandendo un ricciolo, ci si rivela, tuttavia, un'altra scena: esso è costituito da coppie di spirali unite da ponti di filigrana. Un ponte rivela all'ingrandimento la presenza di due riccioli che spuntano dal suo centro. Al centro di questo centro, per così dire, si trova un ponte a 4 direzioni con altri quattro riccioli, al centro dei quali si trova un'altra versione dell'insieme di Mandelbrot.

Quello della versione ingrandita non è proprio lo stesso insieme di Mandelbrot. Continuando a zoomare, sembra che riappaiano gli stessi oggetti, ma un'osservazione più accurata rivela sempre le differenze. La cosa procede così all'infinito, infinitamente varia e spaventosamente bella.

Descriverò due programmi per calcolatore che esplorano gli effetti di operazioni ripetute come quella che porta all'insieme di Mandelbrot. Il primo programma ha generato le illustrazioni a colori che compaiono in queste pagine ed è adattabile per calcolatori personali che abbiano hardware e software adeguati per la grafica. Creerà immagini soddisfacenti anche se si dispone solo di una unità video monocromatica. Il secondo programma è per i lettori che, come me, hanno bisogno di abbandonare per un po' la complessità dell'infinito per rifugiarsi nell'evidente semplicità del finito.

La parola «complesso» qui viene usata in due significati: il significato comune è ovviamente adeguato per descrivere l'insieme di Mandelbrot, ma la parola ha anche un secondo significato più tecnico. Un numero si dice complesso quando è costituito di due parti, che, per ragioni storiche, si chiamano parte reale e parte immaginaria. Questi due termini non hanno più alcun significato specifico: le due parti di un numero complesso si potrebbero anche chiamare Humpty e Dumpty. $7 + 4i$ è un numero complesso con parte reale 7 (Humpty) e parte immaginaria $4i$ (Dumpty). La i in corsivo vicino al 4 indica quale parte del numero complesso è immaginaria.

Ogni numero complesso può essere rappresentato da un punto sul piano e il piano dei numeri complessi è chiamato piano complesso. Per trovare $7 + 4i$, si parte dal numero complesso 0, o $0 + 0i$, e si misurano sette unità a est e quattro unità a nord: il punto risultante rappresenta $7 + 4i$. Il piano complesso è composto da un'infinità non numerabile di numeri di questo genere. Le loro parti reali e quelle immaginarie possono essere positive o negative e possono essere numeri interi o sviluppi decimali.

Sommare o moltiplicare due numeri complessi è facile. Per sommare $3 - 2i$ e $7 + 4i$, si sommano le parti separatamente e il risultato è $10 + 2i$. Moltiplicare numeri complessi è solo un po' più difficile. Per esempio, se si tratta il simbolo i come la x dell'algebra, il prodotto di $3 - 2i$ e $7 + 4i$ è $21 + 12i - 14i - 8i^2$. A questo punto bisogna mettere in gioco una particolare proprietà del simbolo i: i^2 è uguale a -1. Il prodotto può essere quindi semplificato raccogliendo le parti reali e immaginarie: $29 - 2i$.

Ora è possibile descrivere il processo iterativo che genera l'insieme di Mandelbrot. Si inizia con l'espressione algebrica $z^2 + c$, dove z è un numero complesso che può variare e c è un numero complesso prefissato. Poniamo inizialmente z uguale al numero complesso 0. Il quadrato di z è allora 0 e il risultato della somma di c con z^2 è semplicemente c. Sostituiamo ora questo risultato a z nell'espressione $z^2 + c$. La nuova somma è $c^2 + c$. Sostituiamo di nuovo z e la somma successiva è $(c^2 + c)^2 + c$. Continuiamo il procedimento, ponendo sempre come ingresso per il nuovo passo il risultato del passo precedente.

Avvengono strane cose quando le iterazioni riguardano particolari valori di c. Ecco quello che avviene, per esempio, quando c è $1 + i$:

prima iterazione, $1 + 3i$
seconda iterazione, $-7 + 7i$
terza iterazione, $1 - 97i$

Si noti che le parti reali e quelle immaginarie possono crescere, diminuire o cambiare segno. Se questo processo continua, i numeri complessi che ne risultano diventano progressivamente più grandi.

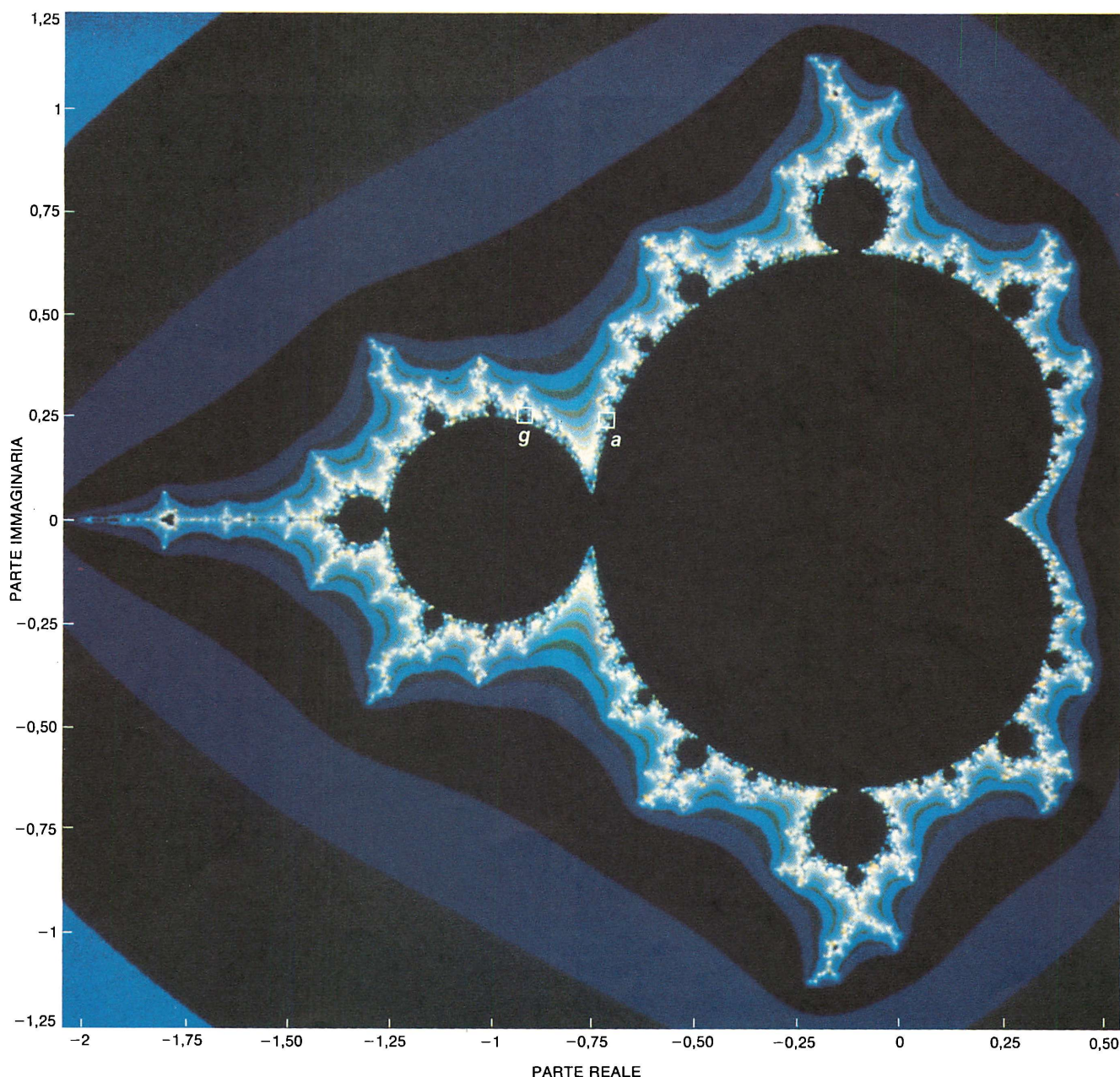
Che cosa si intende, esattamente, per dimensione di un numero complesso? Dato che i numeri complessi corrispondono a punti del piano, possiamo far intervenire il concetto di distanza. La dimensione di un numero complesso non è altro che la sua distanza dal numero complesso 0. Quella distanza è l'ipotenusa di un triangolo rettangolo i cui lati sono la parte reale e quella immaginaria del numero complesso. Per trovare la dimensione del numero complesso, quindi, bisogna elevare al quadrato le sue parti, sommare i due quadrati ed estrarre la radice quadrata della somma. Per esempio, la dimensione del numero complesso $7 + 4i$ è la radice quadrata di $7^2 + 4^2$, ossia circa 8,062. Quando i numeri complessi raggiungono una certa dimensione, nel corso del processo iterativo che ho appena descritto, crescono molto rapidamente: dopo poche altre

iterazioni superano la capacità di qualsiasi calcolatore.

Fortunatamente, posso ignorare tutti i numeri complessi c che corrono all'impazzata verso l'infinito. L'insieme di Mandelbrot è l'insieme di tutti i numeri complessi c per i quali la dimensione di $z^2 + c$ è finita anche dopo un numero indefinitamente grande di iterazioni. Il programma che sto per descrivere ricerca per l'appunto questi numeri. Sono debitore per tutto questo a John H. Hubbard, un matematico della Cornell University, che è un'autorità per quanto riguarda l'insieme di Mandelbrot ed è stato uno dei primi a costruirne un'immagine al calcolatore. La maggior parte delle immagini di questo articolo sono state prodotte da Heinz-Otto Peitgen e dai suoi colleghi dell'Università di Brema, utilizzando gli insegnamenti impartiti a Peitgen da Hubbard.

Il programma di Hubbard è servito da ispirazione per un programma che io chiamo MANDELZOOM. Il programma predispone una matrice detta *fig*, necessaria per salvare le figure. Le entrate di *fig* sono elementi d'immagine distinti, chiamati pixel, disposti secondo una configurazione a griglia. La matrice di Hubbard ha 400 colonne e 400 righe e quella di Peitgen è ancora più grande. I lettori che vogliano adattare MANDELZOOM per uso personale devono scegliere una matrice adeguata alla loro apparecchiatura e al loro carattere. Più grandi sono le matrici più lunghi sono i tempi d'attesa per le immagini, ma migliore è la risoluzione.

Nella prima parte di MANDELZOOM si deve scegliere la regione quadrata del piano complesso da esaminare. L'angolo sudovest del quadrato va indicato con il numero complesso a cui corrisponde. Due variabili del programma, *angoloa* e *angolob*, consentono di introdurre, rispettivamente, la parte reale e quella immaginaria del numero. Va poi specificata la lunghezza di ciascun lato del quadrato introducendo un valore per una variabile detta *lato*.

La seconda parte del programma mette in corrispondenza la matrice *fig* con il

L'insieme di Mandelbrot e le sue coordinate nel piano complesso. I riquadri bianchi indicano i particolari che si vedono nelle illustrazioni delle pagine successive

quadrato che interessa, calcolando la dimensione di una variabile detta *gap*, che indica la distanza all'interno del quadrato tra pixel adiacenti. Per ottenere *gap*, si divide *lato* per il numero di righe (o colonne) di *fig*.

La terza parte costituisce il cuore del programma. Viene effettuata una ricerca dei numeri complessi c dell'insieme di Mandelbrot e vengono assegnati colori ai numeri che sono, in un particolare senso, vicini. Il procedimento deve aver luogo per ogni pixel; la matrice 400 per 400 di Hubbard, quindi, richiede 160 000 calcoli separati. Poniamo che il programma stia attualmente lavorando sul pixel della riga m e della colonna n; la terza parte si divide in quattro passi:

1. Calcolare un numero complesso c che si presuppone rappresenti il pixel: sommare $n \times gap$ ad *angoloa* per ottenere la parte reale *ac* di c; sommare $m \times gap$ ad *angolob* per ottenere la parte immaginaria *bc* di c. Non è necessario includere nel programma il numero immaginario i.

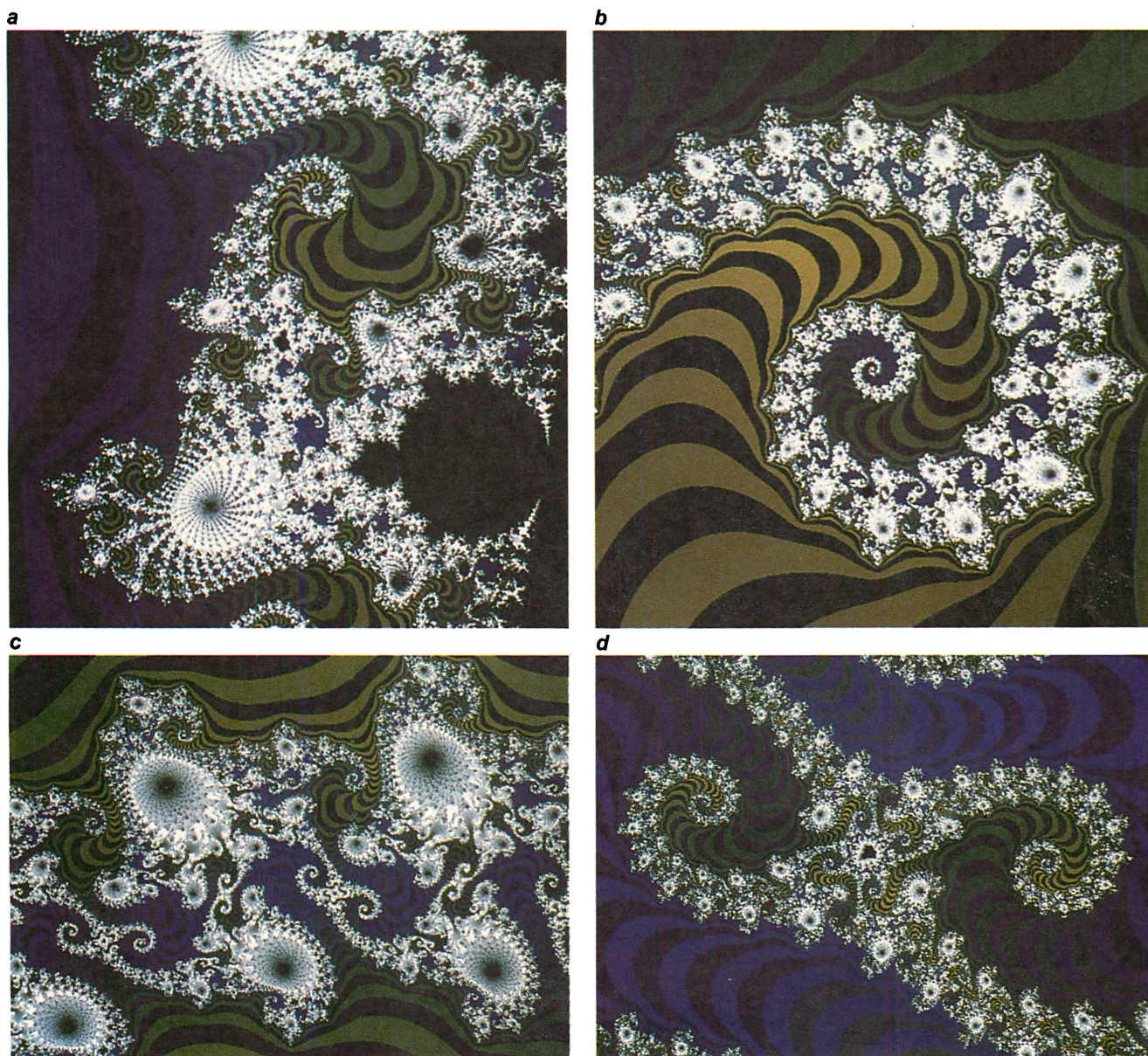
2. Porre inizialmente uguale a $0 + 0i$ una variabile complessa z (con parti *az* e *bz*). Porre uguale a 0 una variabile intera detta *contatore*.

3. Compiere ripetutamente i seguenti tre passi finché la dimensione di z supera 2 oppure la dimensione di *contatore* supera 1000:

$$\begin{aligned} z &\leftarrow z^2 + c \\ \textit{contatore} &\leftarrow \textit{contatore} + 1 \\ \textit{dimensione} &\leftarrow \text{dimensione di } z \end{aligned}$$

Perché il numero 2 è così importante? La teoria delle iterazioni di numeri complessi garantisce che le iterazioni porteranno z all'infinito se e solo se in qualche stadio z raggiunge una dimensione pari o superiore a 2. Risulta che un numero relativamente grande di punti con destino infinito raggiunge 2 dopo solo poche iterazioni. I loro cugini più lenti diventano sempre più rari a valori superiori della variabile *contatore*.

4. Assegnare un colore a *fig (m, n)*, secondo il valore raggiunto da *contatore* alla fine del passo 3. Mostrare sullo schermo il colore del pixel corrispondente. Si noti che il colore di un pixel dipende solo da un numero complesso all'interno del suo piccolo dominio, vale a dire quello al suo angolo nordest; il comportamento di questo numero rappresenta allora il comportamento dell'intero pixel.

Lo schema per l'assegnazione dei colori richiede che il dominio dei valori di *contatore* raggiunti nella matrice sia raggruppato in sottodomini, uno per ogni colore. I pixel per i quali la dimensione

Successivi ingrandimenti del «bastone da pastore» nella regione* a *dell'immagine di pagina 103

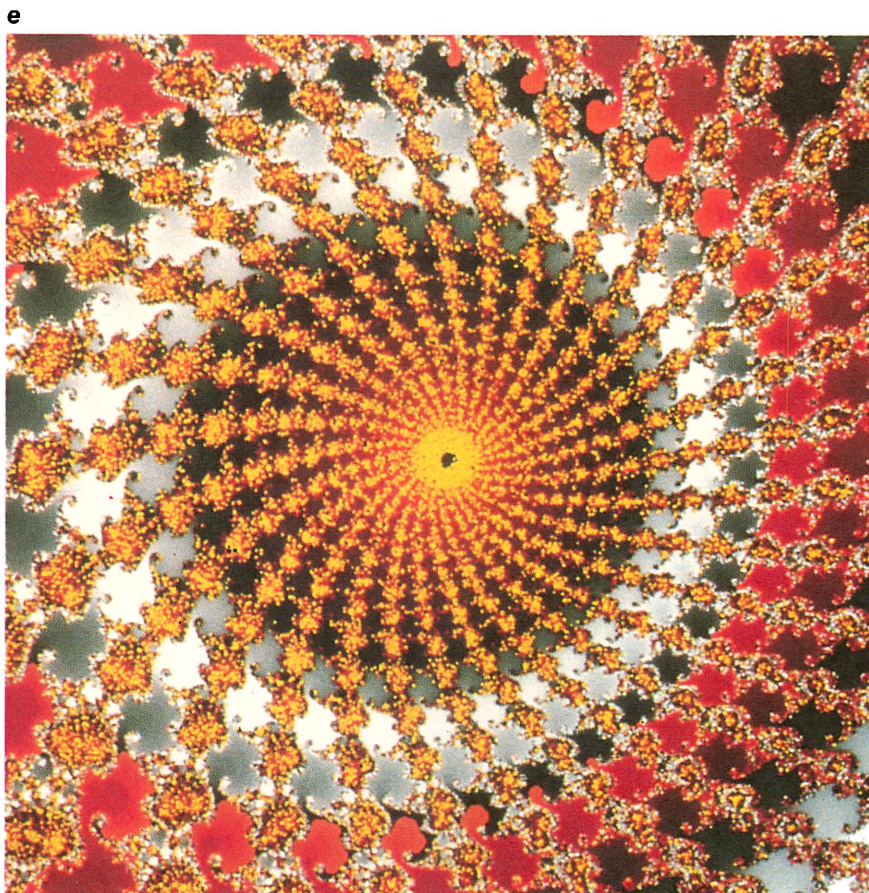
di z raggiunge 2 dopo solo poche iterazioni sono in rosso. I pixel per i quali la dimensione di z raggiunge 2 dopo un numero relativamente elevato di iterazioni sono in violetto, all'estremità opposta dello spettro. I pixel per i quali la dimensione di z è inferiore a 2 anche dopo 1000 iterazioni si presuppone che stiano nell'insieme di Mandelbrot e sono in nero.

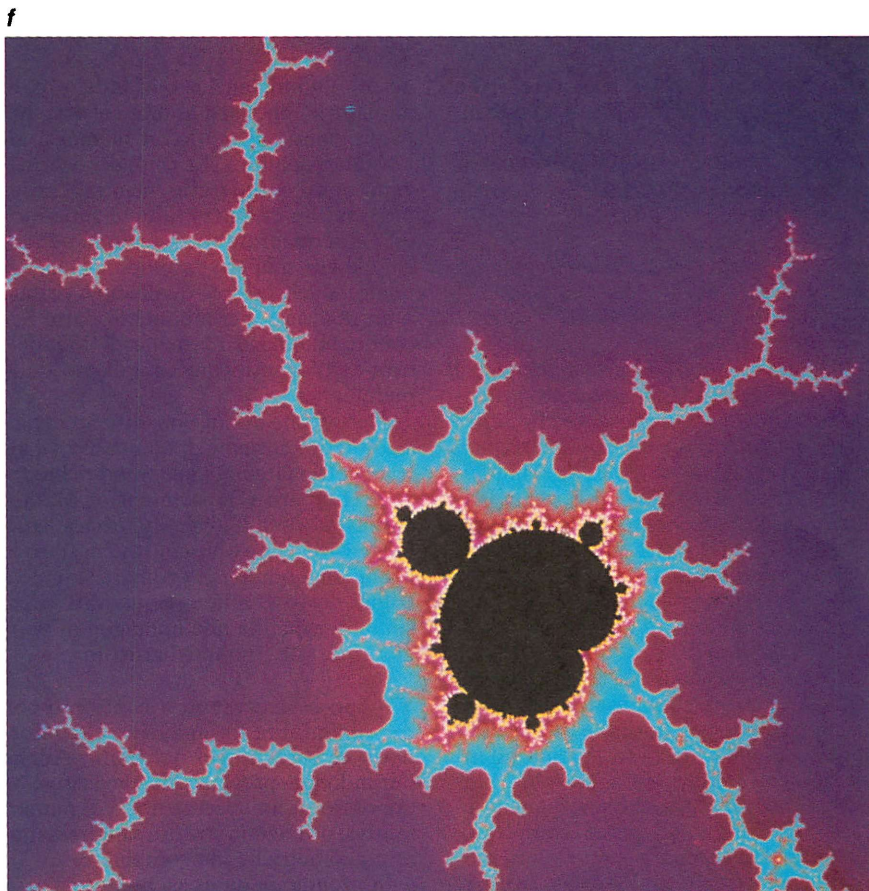
È sensato lasciare non specificati i colori finché non si è determinato il dominio dei valori di *contatore* in un particolare quadrato. Se il dominio è piccolo, si può assegnare l'intero spettro di colore all'interno di quel dominio. Hubbard propone che nel passo 4 si assegni solo il valore di *contatore* a ogni elemento della matrice *fig*. Un programma separato può poi esplorare la matrice, determinare i valori alti e bassi di *contatore* e assegnare di conseguenza lo spettro. I lettori che vogliano seguire questa strada troveranno certo schemi adatti.

Il lettore che non disponga di un monitor a colori può lavorare anche in bianco e nero. I numeri complessi per i quali z è maggiore di 2 dopo r iterazioni sono in bianco; gli altri sono in nero. r può essere scelto a proprio piacimento. Per evitare di dover far girare il programma per intere notti la matrice può essere, diciamo, di 100 righe per 100 colonne. Secondo Hubbard è del tutto ragionevole ridurre il numero massimo di iterazioni per punto da 1000 a 100. Il risultato di un programma del genere è una suggestiva immagine a punti della sua controparte in colore (*si veda l'illustrazione a pagina 106*).

Che potenza ha la «lente zoom» di un calcolatore personale? In qualche misura dipende dall'effettiva dimensione dei numeri che la macchina può manipolare. Per esempio, secondo Magi (il mio amanuense dei microcalcolatori alla University of Western Ontario), il PC IBM utilizza il microelaboratore 8088, un chip prodotto dalla Intel Corporation e progettato per manipolare numeri a 16 bit. Con il procedimento detto della doppia precisione è possibile portare la lunghezza dei numeri a 32 bit. Con questa doppia precisione, Magi e io abbiamo calcolato che si possono realizzare ingrandimenti dell'ordine di 100 000 volte. Software di maggior precisione, che permette in effetti di concatenare questi gruppi di bit, può far arrivare la precisione numerica a centinaia di cifre significative. L'ingrandimento dell'insieme di Mandelbrot che si può teoricamente raggiungere è molto maggiore dell'ingrandimento necessario per risolvere il nucleo dell'atomo.

Dove si dovrebbe esplorare il piano complesso? Vicino all'insieme di Mandelbrot, naturalmente, ma dove precisamente? Hubbard afferma che «ci sono zilioni di bellissimi punti». Come un turista in una terra di infinita bellezza, trabocca di suggerimenti sui posti che i let-

e

Un occhio composto che spunta da una regione dell'immagine* d *della pagina a fronte

f

Un insieme di Mandelbrot in miniatura nella regione* f *dell'immagine a pagina 103, collegato all'insieme principale da un filamento

Mandelbrot in miniatura a punti generato da un monitor monocromatico

tori potrebbero voler esplorare. Non hanno nomi come Hong Kong o Isfahan: «Provate con l'area che ha parte reale compresa tra 0,26 e 0,27 e parte immaginaria compresa tra 0 e 0,01». Propone anche altre due località:

Parte reale	*Parte immaginaria*
da −0,76 a −0,74	da 0,01 a 0,03
da −1,26 a −1,24	da 0,01 a 0,03

Il lettore che esamini le immagini a colori che accompagnano questo articolo dovrà tenere ben presente che tutti i punti di colore diverso dal nero non appartengono all'insieme di Mandelbrot. La bellezza sta in gran parte nell'alone di colori assegnati ai punti in fuga. In effetti, se si dovesse vedere l'insieme isolato, la sua immagine non sarebbe forse così piacevole: l'insieme è tutto coperto di filamenti e di versioni in miniatura di se stesso.

In realtà, nessuno dei Mandelbrot in miniatura è una copia esatta dell'insieme genitore e nessuno di essi è esattamente uguale a un altro. Vicino all'insieme genitore ci sono ancor più Mandelbrot in miniatura, che sembrano liberamente sospesi nel piano complesso. L'apparenza è però ingannevole. Hubbard e un suo collega, Adrian Donady dell'Università di Parigi, hanno dimostrato un sorprendente teorema secondo il quale l'insieme di Mandelbrot è connesso. Quindi anche i Mandelbrot in miniatura, che sembrano sospesi nel piano, sono collegati con filamenti all'insieme genitore. Le miniature si trovano quasi dappertutto vicino all'insieme genitore e sono di tutte le dimensioni. Ogni quadrato della regione ne racchiude un numero infinito, di cui nel migliore dei casi solo qualcuno è visibile a un qualsiasi ingrandimento dato. Secondo Hubbard, l'insieme di Mandelbrot è «l'oggetto più complicato della matematica».

I lettori che desiderino altre immagini a colori dell'insieme di Mandelbrot e altri oggetti matematici possono richiedere a Hubbard (Department of Mathematics, Cornell University, Ithaca, New York 14853) un opuscolo in cui si trova anche un modulo per ordinare stampe a colori di 40 centimetri quadrati circa, simili per qualità alle immagini di Peitgen riportate in questo articolo.

Dopo essere venuti a confronto con un'infinita complessità è confortante cercare rifugio nel finito. Anche iterando un processo di elevazione al quadrato su un insieme finito di numeri interi si ottengono strutture interessanti, non geometriche ma combinatorie.

Si prenda un qualsiasi numero a caso compreso tra 0 e 99. Lo si elevi al quadrato e si estraggano dal risultato le ultime due cifre, che a loro volta debbono darci un numero compreso tra 0 e 99. Per esempio, 59^2 è uguale a 3481: le ultime due cifre sono 81. Ripetete il procedimento e presto o tardi genererete un numero che avete già incontrato. Per esempio 81 porta alla successione 61, 21, 41 e 81, poi questa successione di quattro numeri si ripete all'infinito. Questi cicli nascono sempre da processi iterativi su insiemi finiti. In effetti è facile rendersi conto che ci deve essere almeno un numero che viene ripetuto dopo 100 operazioni in un insieme di 100 numeri; il primo numero ripetuto porta quindi a un ciclo. C'è un bel programma, per individuare i cicli, che non richiede quasi memoria, ma ne parleremo più avanti.

Ci vuole soltanto un'ora per rappresentare con un diagramma i risultati della procedura di elevazione al quadrato. Si rappresenti ciascun numero da 0 a 99 con un punto distinto su un foglio di carta. Se la procedura di elevazione al quadrato porta da un numero a un nuovo numero, si congiungano i due punti con una freccia. Per esempio una freccia andrebbe dal punto 59 al punto 81. Le prime connessioni nel diagramma potrebbero portare a cicli intrecciati; è pertanto una buona idea ridisegnare le frecce di tanto in tanto in modo che non ce ne siano due che si incrociano. Un diagramma di iterazione senza incroci è sempre possibile.

Ci si può spingere anche più in là. Spesso si hanno sottodiagrammi separati, che si possono visualizzare in un modo che illumina alcune delle simmetrie che nascono dalle iterazioni. Per esempio, il diagramma di iterazione senza incroci relativo alla procedura di elevazione al quadrato degli interi da 0 a 99 comprende sei sottodiagrammi non connessi. I sottodiagrammi si presentano in coppie identiche e ciascuno di essi è fortemente simmetrico *(si veda la figura della pagina a fronte)*. Il lettore è in grado di spiegare la simmetria? Che cosa accadrebbe nel caso si usassero gli interi da 0 a 119? C'è una relazione tra il numero di sottodiagrammi non connessi che si trovano nel diagramma e l'intero più grande della successione?

Schemi di iterazione analoghi valgono per alcuni dei numeri complessi dell'insieme di Mandelbrot: per certi valori di c, iterazioni ripetute di $z^2 + c$ possono portare a un ciclo finito di numeri complessi. Per esempio, il numero complesso $0 + 1i$ porta a una oscillazione indefinita tra i due numeri complessi $-1 + 1i$ e $0 - 1i$. Il ciclo può persino avere un solo membro. Che si trovino in un insieme finito o nell'insieme infinito di Mandelbrot, questi cicli si chiamano attrattori.

Ciascuno dei sei sottodiagrammi del diagramma di iterazione per gli interi da 0 a 99 comprende un attrattore. Geometricamente, un attrattore può essere rappresentato come un poligono; gli insiemi di numeri che portano a esso sono rappresentabili come alberi.

Un modo per trovare un attrattore col calcolatore consiste nell'immagazzinare ogni nuovo numero generato in una matrice appositamente costituita. Si confronta il nuovo numero con tutti i numeri precedentemente immagazzinati nella matrice e, se si trova una corrispondenza, si stampano tutti i numeri della matrice dal numero corrispondente a quello appena creato. Il metodo è chiaro e facile da programmare, ma può richiedere molto tempo se la matrice è grande. Per scoprire un ciclo attrattore in una matrice formata da n numeri ci vorrebbero qualcosa come n^2 confronti: ogni nuovo numero arriva a dover essere confrontato con n numeri della matrice.

C'è un programmino ingegnoso che troverà un attrattore molto più velocemente. Il programma non richiede n parole di memoria ma solo due e può essere codificato sulla più semplice delle calcolatrici tascabili programmabili. Lo si può trovare in un interessante libro intitolato *Mathematical Recreations for the Programmable Calculator,* di Dean Hoffman della Auburn University e Lee Mohler dell'Università dell'Alabama. Inutile dire che molti degli argomenti trattati nel libro sono facilmente trasformabili in programmi per calcolatori.

Il programma si chiama RHOP perché la successione di numeri che infine si ripete assomiglia a un pezzo di corda [*rope* in inglese] con un cappio a un'estremità e perché assomiglia anche alla lettera greca rho (ϱ). Nel programma vi sono due variabili dette *lento* e *veloce* e a entrambe viene inizialmente attribuito il valore del numero di partenza. Il ciclo iterativo del programma comprende appena tre istruzioni:

$$\begin{aligned} veloce &\leftarrow veloce \times veloce \pmod{100} \\ veloce &\leftarrow veloce \times veloce \pmod{100} \\ lento &\leftarrow lento \times lento \pmod{100} \end{aligned}$$

L'operazione mod 100 estrae le ultime due cifre dei prodotti. Si noti che l'elevazione al quadrato è effettuata due volte sul numero *veloce* ma solo una volta sul numero *lento. Veloce* compie il tragitto dalla coda alla testa due volte più veloce di *lento.* Nella testa, *veloce* riprende *lento* quando *lento* ha compiuto mezzo giro. Il programma esce dal suo ciclo iterativo quando *veloce* è uguale a *lento.*

L'attrattore viene identificato ripetendo la procedura di elevazione al quadrato per il numero attualmente assegnato a *lento.* Quando il numero torna a presentarsi, si ferma il programma e si stampa la successione di numeri che si frappone tra le due occorrenze.

I sei componenti del diagramma di iterazione dell'elevazione al quadrato per i primi 100 interi

Montagne frattali, piante graftali e altra grafica al calcolatore della Pixar

di A. K. Dewdney

Le Scienze, febbraio 1987

Posso facilmente immaginare il primo vero lungometraggio generato dal calcolatore. È l'anno 1991. Inciampo camminando nel corridoio tra le poltrone con in mano uno smisurato cartoccio di popcorn sintetico e una bibita contenente alcuni additivi che rendono inutili i normali ingredienti. Le luci si abbassano, il sipario si apre e lo schermo argenteo si anima con una riduzione della trilogia di J. R. R. Tolkien *Il signore degli anelli*. Lo hobbit Frodo passeggia per una stretta valle. In lontananza, i picchi innevati di una montagna si alzano frastagliati verso il cielo. In primo piano, alberi esotici e piante di specie sconosciute brillano alla luce del sole. La scena cambia e compare un mago che scruta in una sfera di cristallo. Al centro della sfera si vede una fortezza con gli spalti merlati avvolti dalle fiamme.

È difficile dire con esattezza quanto potrà essere convincente in questo film Frodo che cammina e parla, ma sono sicuro che le montagne, le piante, la sfera di cristallo e le fiamme riusciranno tutte magnificamente. Il successo sarà dovuto ampiamente al software e all'hardware avveniristici di una società di nome Pixar, già Lucasfilm Computer Graphics Laboratory. Dopo aver visitato questo affascinante centro della grafica al calcolatore di San Rafael in California, posso mettere a parte i lettori dei più riposti segreti di montagne e alberi. Chiunque possieda un calcolatore domestico è ora in grado di generare immagini che assomigliano molto a questi oggetti. La limitazione di spazio che la rubrica impone mi impedisce di trattare estesamente della sfera di cristallo e del fuoco; svelerò, però, i principi di base per generarli.

Nell'ipotetico film descritto, la telecamera potrebbe zoomare sulle cime innevate alle spalle di Frodo. Non si potrebbe vedere un ammasso più spaventoso: ogni vetta è formata da vette più piccole e così via: un regresso all'infinito di piccoli picchi. Perfino un Orco, quella bestia mostruosa dai piedi di cuoio, non si troverebbe a proprio agio su quegli scabri pendii.

In linea di principio è facile generare una catena montuosa di questo aspetto. Per semplificare, ammettiamo che il terreno copra un'area triangolare. Si suddivide allora il triangolo in quattro triangoli più piccoli, trovando il punto di mezzo di ciascun lato e congiungendo i nuovi punti con tre segmenti. Ciascun triangolo viene a sua volta suddiviso nella stessa maniera. Si continua il procedimento fino a raggiungere i limiti di risoluzione o del tempo di calcolo. Il risultato - un reticolo di triangoli piuttosto monotono - può essere ravvivato aggiungendovi un po' di movimento in verticale: ogni volta che si aggiunge alla scena un nuovo punto medio, lo si sposta verso l'alto o verso il basso di una misura casuale. Gli spostamenti casuali, che in generale devono essere ridotti a mano a mano che i triangoli diventano più piccoli, trasformano i triangoli in vette frastagliate che si alternano a valli (*si veda l'illustrazione in alto a pagina 110*).

Perché questa tecnica dovrebbe produrre montagne dall'aspetto naturale? La risposta può risiedere in parte nel fatto che il procedimento produce un frattale: un tipo di oggetto che crescendo rivela un maggior numero di dettagli. A quanto pare, in tutta la natura si possono vedere frattali. Benoit B. Mandelbrot, l'infaticabile studioso di frattali del Thomas J. Watson Research Center della IBM a Yorktown Heights, New York, utilizza linee costiere per illustrare l'idea di fondo. Immaginiamo che ci sia chiesto di misurare la costa francese con un'asta lunga un chilometro. Facendo ruotare l'asta sulle sue estremità, con una faticosa marcia lungo la costa si arriva a calcolare il numero di chilometri. Molte piccole baie e molti piccoli promontori, però, vengono tralasciati e la lunghezza finale misurata in questo modo non è del tutto esatta. Se si ripete l'esercizio con un righello da un metro si ottiene una misura più precisa e più lunga. Anche in questo caso, però, viene trascurato un gran numero di minuscole insenature e lingue di terra. Senza dubbio, un righello da un centimetro sarebbe più preciso.

Come regola generale possiamo dire che la lunghezza della costa misurata aumenta con il ridursi dell'asta di misurazione. La relazione tra la lunghezza misurata e la dimensione dell'asta è un particolare numero detto «dimensione frattale». A differenza di una comune dimensione, una dimensione frattale di solito è espressa sotto forma di frazione, non di numero intero. La linea costiera in questione potrebbe avere, per esempio, dimensione frattale pari a 3/2. Questa forma può essere vista come una via di mezzo tra una forma a una dimensione (una linea retta) e una forma a due dimensioni (un piano). Se una linea costiera fosse relativamente diritta, la sua dimensione frattale sarebbe vicina a 1; se invece fosse molto frastagliata, la sua dimensione frattale si avvicinerebbe a 2 come se cercasse di riempire un piano a due dimensioni.

Il modello frattale della natura implica un regresso infinito di dettagli. Dal punto di vista della grafica al calcolatore, la questione del regresso all'infinito non si pone; è sufficiente che il paesaggio appaia dettagliato a tutti i livelli di ingrandimento. Fino ai limiti di risoluzione dello schermo, le montagne da generare hanno caratteristiche con finezza pari ai triangoli finali usati nella suddivisione descritta in precedenza. Anche se l'algoritmo completo per disegnare montagne è troppo lungo e complesso per poterlo descrivere in questa sede, c'è un semplice programma chiamato MOUNTAIN che disegna il monte Mandelbrot in sezione trasversale. MOUNTAIN illustra l'idea fondamentale di punti di suddivisione a spostamento casuale lungo un asse verticale. L'artista frattale inizia con un unico segmento orizzontale. Si determina il punto di mezzo e lo si sposta su o giù di una misura casuale. Ciascuno dei due segmenti risultanti viene poi suddiviso e perturbato. Il procedimento può essere proseguito in maniera analoga alla tecnica di suddivisione dei triangoli.

MOUNTAIN conserva due matrici, dette *punti* e *linee*, per seguire il profilo montuoso. Ciascuna matrice ha due colonne e un numero di righe adeguato alla risoluzione dello schermo (per esempio 2048). Le due colonne di *punti* contengono coordinate e le due colonne di *linee* contengono indici; ciascuna linea è specificata come coppia di posizioni nella matrice *punti* che designa le coordinate delle estremità della linea. Dato che è interessante osservare come le suddivisioni che si susseguono formino il profilo di una montagna a partire da un poligono apparentemente poco promettente, MOUNTAIN mette ogni generazione sotto il controllo dell'utente. Al termine di un singolo ciclo principale, il programma chiede all'utente se vuole un'altra iterazione. Se la risposta è positiva, l'esecuzione torna all'inizio del programma.

Il ciclo principale trasforma gli insiemi attuali di punti e linee in nuovi insiemi grandi il doppio. Per ottenere questo risultato scorre *linee* una riga alla volta, cerca gli indici dei punti corrispondenti e richiama le loro coordinate da *punti*. Con le coordinate delle estremità di una data linea, il programma calcola le coordinate del punto di mezzo della linea stessa, modificando casualmente, nel

corso del procedimento, la coordinata *y*. L'algoritmo che segue fornisce una base adeguata per un programma. Le variabili *j* e *k* indicano le righe di *punti* e *linee* che a un dato momento vengono riempite con gli ultimi risultati della suddivisione. Le variabili *pt* e *ln* registrano il numero di punti e il numero di linee che formano la montagna prima che il programma entri nel ciclo principale. All'inizio *j* è uguale a *pt* e *k* è uguale a *ln*. L'indice *i* va da 1 a *ln*.

```
j  ← j + 1
k  ← k + 1
a  ← linee (i,1)
b  ← linee (i,2)
x1← punti (a,1)
y1← punti (a,2)
x2← punti (b,1)
y2← punti (b,2)
punti (j,1) ← (x1 + x2)/2
punti (j,2) ← (y1 + y2)/2 +
                   a caso (intervallo)
linee (i,2) ← j
linee (k,1) ← j
linee (k,2) ← b
```

Questa parte di MOUNTAIN si spiega in gran parte da sé. Una volta calcolate le coordinate del *j*-esimo punto, l'indice *j* viene memorizzato come secondo punto dell'*i*-esima linea e primo punto della *k*-esima linea. Il primo punto della linea *i*-esima è lo stesso di prima e il secondo punto della linea *k*-esima è identico al secondo punto originario della linea *i*-esima, vale a dire quello con indice *b*.

Al termine del ciclo di calcolo, *pt* e *ln* devono essere riportati, rispettivamente, agli ultimi valori di *j* e di *k*. Alla variabile *intervallo* inizialmente l'utente dà come valore la massima quantità di casualità verticale che può essere data al punto di suddivisione. Ogni volta che il ciclo viene completato, questa variabile deve essere divisa per 2 in modo che le fluttuazioni casuali siano sempre in scala con la dimensione delle caratteristiche da variare. La funzione *a caso* (*intervallo*) intende esprimere la selezione di un numero casuale compreso tra 0 e il valore di *intervallo* (al momento dato).

Se appaiono efficaci le montagne che stanno alle spalle di Frodo, gli alberi e le piante che lo circondano non sono da meno. Sono allo stesso tempo realistici e fiabeschi. Sembrano reali perché hanno ramificazioni simili a quelle delle vere piante, e fiabeschi perché non appartengono a specie familiari; il progettista grafico ha a disposizione un tale numero di parametri da non poter resistere alla tentazione di creare qualcosa di nuovo.

Le nuove «specie» sono denominate piante graftali, perché sono basate su grafi e hanno un'implicita natura frattale. Per implicita natura frattale intendo il fatto che le regole per generare la topologia di base delle piante potrebbero essere (ma non lo sono) applicate al limite di risoluzione dello schermo. In breve, un ramo non si sviluppa in un regresso all'infinito di rametti. Una volta sviluppato, il grafo che costituisce la base di una pianta può essere trasformato in una miriade di specie convincenti se lo si interpreta in termini di dimensione, colore, spessore, struttura e così via.

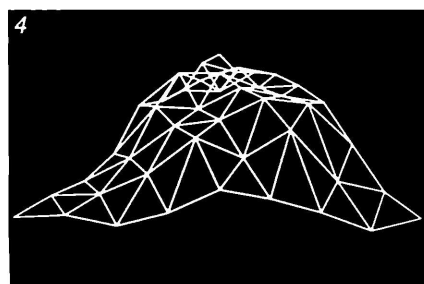
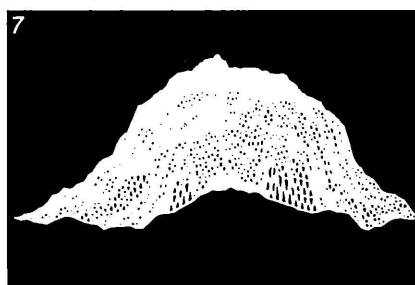
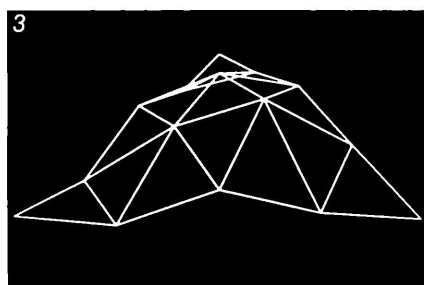
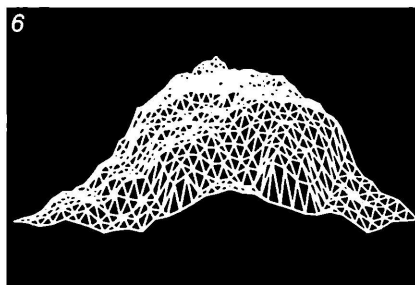
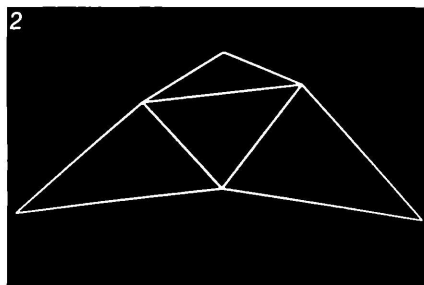
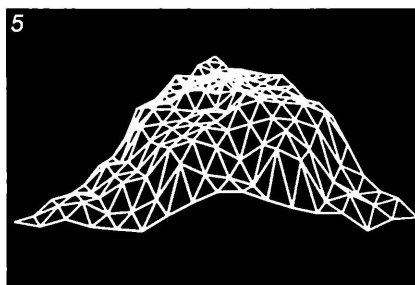
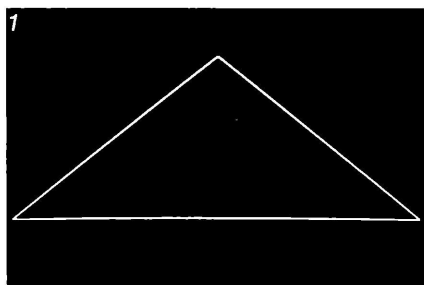
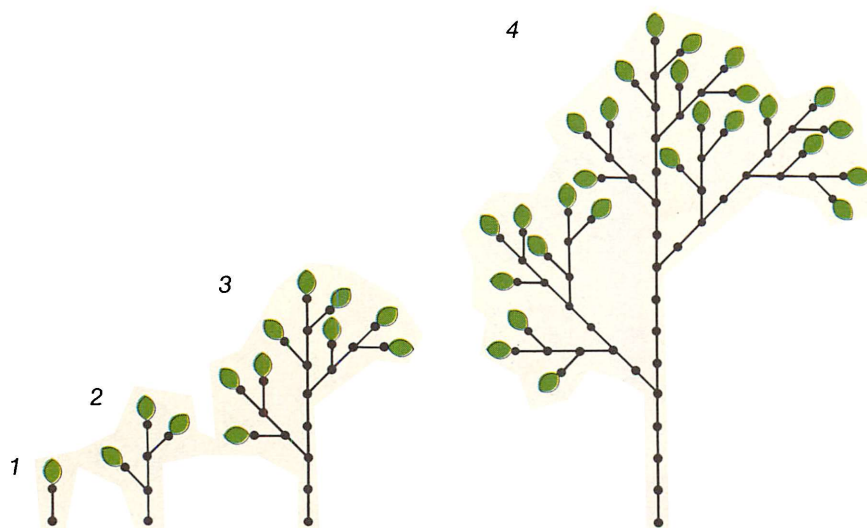
I grafi che sottendono una data pianta sono prodotti da sistemi *L*, una classe di grammatiche introdotta nel 1968 dal biologo e matematico danese Aristid Lindenmeyer. Un sistema *L* è in sostanza un insieme di regole per derivare nuove stringhe di simboli da vecchie stringhe. Le regole comprendono sequenze di sostituzione di simboli per singoli simboli. Per esempio, usando i numeri 0 e 1 e i simboli di parentesi quadra aperta e chiusa si può generare una vasta gamma di forme botaniche complesse con le seguenti regole:

0 → 1 [0]1[0]0
i → 1 1
[→ [
] →]

Per vedere come funzionano le regole, supponiamo di partire con la stringa formata da un unico 0. A ogni simbolo di sinistra della stringa si sostituisce il suo corrispondente simbolo di destra in modo da ottenere le seguenti stringhe in successione:

0
1[0]1[0]0
1 1[1[0]1[0]0]1 1[1[0]1[0]0]1[0]1[0]0

Queste stringhe possono essere trasformate in grafi a forma di albero trattando ogni numero (0 o 1) come un segmento

Montagne frattali generate dal calcolatore imitano la natura

La suddivisione di triangoli produce una montagna

Quattro generazioni di una pianta graftale

e ogni parentesi come un punto di ramificazione. I segmenti 0 e i segmenti 1 sono di uguale lunghezza; tipicamente si distinguono perché si lasciano spogli tutti i segmenti 1, mentre si pone una foglia all'estremità esterna di ogni segmento 0.

Per esempio, lo stelo (o tronco) della stringa 1[0]1[0]0 è formato dai tre simboli che non si trovano tra parentesi; un segmento 1 è sormontato da un secondo segmento 1 e poi ancora da un segmento 0. Due rami, ciascuno formato da un singolo segmento 0, germinano da questa formula. Il primo ramo ha l'attaccatura sopra il primo segmento e il secondo sopra il secondo segmento. Prima di studiare l'illustrazione in basso a sinistra, i lettori potrebbero divertirsi a disegnare un po' di generazioni della struttura. Per amore di realismo, si possono aggiungere al modello altre caratteristiche interpretative. Si potrebbe specificare che per qualunque tronco dato (indipendentemente dal fatto che sia o meno il tronco principale) i rami dovrebbero dipartirsi alternativamente verso destra e verso sinistra. Non volendo imporre alcunché di più arduo ai lettori che desiderassero programmare graftali, sono lieto di suggerire, per le piante, semplici ramoscelli. I professionisti della Pixar trasformano la grammatica appena descritta in bellissime piante come quelle dell'illustrazione in alto della pagina a fronte.

Un programma in due parti detto PLANT genera l'*n*-esima stringa della successione precedentemente illustrata e poi la presenta come disegno al tratto. Nella sua prima fase, PLANT conserva le stringhe che genera in due matrici di simboli dette *stringaA* e *stringaB*. Ciascuna generazione di piante occupa una delle due matrici in modo alternato: la generazione di una matrice è derivata dalla generazione precedente dell'altra. Non è strettamente necessario immagazzinare simboli in queste matrici: se il programma esegue correttamente le sostituzioni, andranno benissimo anche i numeri 0, 1, 2 e 3.

Le regole del sistema *L* sono inglobate in enunciati condizionali. Per esempio, si può adattare il seguente passo di codice algoritmico per trasformare uno 0 nella *i*-esima posizione di *stringaA* in nove nuovi simboli di *stringaB:*

```
se stringaA(i) = 0, allora
    stringaB(j) ← 1
    stringaB(j + 1) ← 2
    stringaB(j + 2) ← 0
    stringaB(j + 3) ← 3
    stringaB(j + 4) ← 1
    stringaB(j + 5) ← 2
    stringaB(j + 6) ← 0
    stringaB(j + 7) ← 3
    stringaB(j + 8) ← 0
    j ← j + 9
```

Qui 0 e 1 stanno per se stessi, mentre 2 e 3 stanno, rispettivamente, per [e]. Se l'*i*-esimo simbolo di *stringaA* è 0, allora il programma inserisce la successione 1,

2, 0, 3, 1, 2, 0, 3, 0 in nove posizioni successive della matrice *stringaB* a partire dall'indice *j* (la prima posizione della seconda matrice che non è stata ancora riempita). Un unico ciclo nella prima fase di PLANT contiene quattro enunciati condizionali di questo genere, uno per ogni possibile simbolo incontrato. Il ciclo usa l'indice *j* come riferimento al simbolo della generazione che si sta elaborando. Il ciclo viene eseguito per il numero di generazioni voluto dall'utente. A ogni stadio, PLANT può chiedere all'utente se vuole un'altra (più lunga) stringa di simboli.

La seconda fase di PLANT, quella grafica, trasforma in un disegno la stringa prodotta dalla prima fase. L'operazione è compiuta ricorsivamente. Finché non incontra una parentesi sinistra, o 2, disegna una successione di segmenti in una data direzione. Quando viene presa in esame una parentesi sinistra di una data coppia, il programma disegna il successivo segmento in una nuova direzione, spostata di 45 gradi in senso antiorario rispetto a quella precedente. La fine del procedimento è segnalata dalla comparsa della corrispondente parentesi destra; qui può essere disegnata una foglia (di forma e colore completamente affidati alla fantasia del lettore). La comparsa di una seconda parentesi sinistra provoca la ripetizione del procedimento, solo che ora la nuova direzione è di 45 gradi in senso orario. Il resto è automatico.

PLANT utilizza un fattore di scala che dipende dalla complessità della pianta da disegnare. L'n-esima generazione, per esempio, è alta approssimativamente 2^n segmenti. Se lo schermo ha un'altezza di 200 pixel, i segmenti devono essere più corti di $200/2^n$. Senza dubbio, i lettori ambiziosi cercheranno varianti nella grammatica generativa, negli angoli di ramificazione e nella forma delle foglie. Se si eseguono queste varianti sullo stesso schermo, appariranno paesaggi di piante e alberi (non molto realistici, bisogna ammettere).

La sfera di cristallo dell'ipotetico film tratto da Tolkien sarebbe realizzata con una tecnica chiamata tracciamento di raggi (*ray tracing*); i merli in fiamme sarebbero simulati seguendo il movimento di un grosso sistema di particelle.

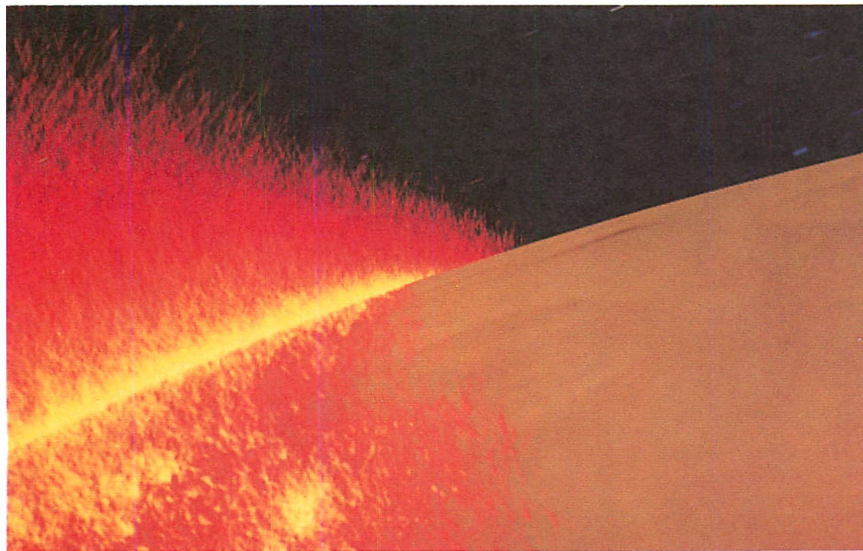
Il tracciamento di raggi richiede di specificare sia la geometria tridimensionale di una scena sia la posizione di una sorgente di luce. Quando lascia una sorgente, la luce si imbarca in una complicata storia di riflessioni e rifrazioni. L'occhio di un osservatore che si trovi sulla scena intercetterà alcuni raggi di luce che fluiscono dalla sorgente ma ne mancherà molti altri, in realtà la maggior parte. Per non sprecare tempo e potenza di calcolo, la tecnica del tracciamento di raggi lavora nella direzione opposta. Immaginiamo per un momento che la luce lasci invece l'occhio. Un ampio fascio di raggi si distende a ventaglio nella scena. Se un raggio colpisce una superficie riflettente o rifrangente, saetta via in una nuova direzione determinata dalle leggi dell'ottica. Infine il raggio colpisce una superficie assorbente, assumendo il colore ivi assegnato. Quel colore è registrato nel pixel corrispondente alla direzione del raggio di partenza.

Piante graftali prodotte alla Pixar

Un'immagine di palle da biliardo generata dal calcolatore illustra la tecnica di tracciamento di raggi

Qui sopra si può vedere un'immagine generata con questa tecnica. I raggi tracciati nella scena composta da palle di biliardo hanno una storia relativamente semplice. Nonostante la semplicità, però, possiamo vedere riflessi l'interno di una sala da gioco e un uomo che, in piedi con una stecca in mano, osserva il colpo.

Il grande sistema di particelle che si potrebbe usare per generare i merli in fiamme è lo sviluppo logico dei piccoli ammassi di punti che rendono simbolicamente le esplosioni in miniatura nei videogiochi. Alla Pixar, però, un sistema di particelle è molto più raffinato. All'interno di una certa regione, moltissime particelle vivono, si muovono e conducono la loro esistenza. Sotto il controllo del calcolatore, ciascuna particella è un punto che si sposta secondo una dinamica predeterminata. Nata in un certo istante, può muoversi per un po', magari dando anche vita a nuove particelle; poi può morire.

Sistemi di particelle sono stati usati in modo spettacolare in una scena del film *Star Trek II*. Una bomba Genesi viene sganciata su un pianeta morto pieno di crateri. La bomba crea un anello di stra-

Una scena dalla sequenza Genesi in **Star Trek II**

ne fiamme scintillanti che alla fine inghiottono il pianeta. Quando infine si spengono, vediamo la superficie trasformata in una lussureggiante biosfera. L'effetto è stato prodotto dalla Pixar nella sua precedente incarnazione come Lucasfilm Computer Graphics Laboratory. L'anello di fuoco in espansione era formato da sistemi di particelle in cui alcune particelle davano origine a interi nuovi sistemi. I nuovi sistemi rappresentavano particelle che, scagliate verso l'alto dalla superficie del pianeta, cambiavano colore e persino ricadevano sotto l'influenza della gravità.

Alvy Ray Smith, che dirige il settore ricerca e sviluppo alla Pixar, mi ha accompagnato in un giro dell'azienda durante la visita a San Rafael. Oltre che con Smith, pioniere nell'applicazione del metodo frattale alla vita delle piante col calcolatore, ho avuto un incontro con Loren Carpenter, specialista di montagne frattali, con Robert L. Cook, esperto del tracciamento di raggi, e con William Reeves, creatore dei sistemi di particelle. Nel bel mezzo di una discussione sul software grafico, Smith mi ha sorpreso affermando che l'attività principale dell'azienda non è tanto la produzione di effetti speciali per Hollywood quanto la costruzione di un calcolatore dedicato alla grafica e chiamato, abbastanza naturalmente, Pixar Image Computer.

Al cuore del Pixar Image Computer c'è una memoria da 24 megabyte per 2000 × 2000 pixel. È una risoluzione più che sufficiente per la maggior parte delle applicazioni. Ciascun pixel, per di più, è rappresentato da 48 bit di memoria, sufficienti a conservare copiose informazioni sul colore e la trasparenza. La grande memoria del Pixar è controllata da quattro elaboratori paralleli ad alta velocità, totalmente programmabili, che possono eseguire circa 40 milioni di istruzioni al secondo, una velocità che è di molti ordini di grandezza superiore a quella dei comuni calcolatori. Un'unità video comunica con la memoria a una velocità di 480 milioni di byte al secondo.

I primi Pixar messi in commercio sono destinati all'elaborazione di immagini in campo medico, al rilevamento a distanza, al disegno tecnico e all'animazione. Forse saranno usati anche per generare il mio ipotetico film.

Ai Laboratori Bell il lavoro è gioco e le malattie dei terminali sono benigne

di A. K. Dewdney

Le Scienze, novembre 1985

Negli AT&T Bell Laboratories di Murray Hill, New Jersey, è impossibile tracciare una netta linea di demarcazione tra lavoro e gioco. È curioso come di tanto in tanto, dal laboratorio, emergano giochi seri dal traboccare di una sorprendente sorgente spontanea di creatività scientifica. Ne sono esempi il terminale Blit e CRABS (granchi), una ricreazione crostacea che a volte complica l'uso del Blit.

Sviluppato alcuni anni fa a Murray Hill da Rob Pike e Bart Locanthi, il Blit (come venne affettuosamente chiamato) era una versione preliminare del nuovo terminale DMD 5620 della Teletype Corporation. Il Blit e la sua incarnazione Teletype sono terminali per la multiprogrammazione.

Durante una visita ai laboratori, un paio di anni fa, venni invitato da Pike a vedere il DMD 5620 in azione. Su terminali di questo tipo (che d'ora in avanti chiamerò semplicemente terminali Blit), lo schermo di visualizzazione può essere diviso in finestre, che a volte si sovrappongono. Ciascuna finestra si comporta come un piccolo schermo autonomo; più specificamente, ciascuna visualizza l'uscita di un programma distinto. I programmi possono girare sul microelaboratore interno del terminale o sul calcolatore ospite del Blit. Una tramatura grigia copre le parti di schermo che non vengono utilizzate.

Tre programmi occupavano le finestre davanti a noi. In una finestra c'era un *editor*, un «redattore» di testi sperimentale che veniva sottoposto a prove. Nella seconda finestra un altro «redattore» mostrava il testo emesso dal primo programma. Nella terza finestra un *debugger*, un programma di messa a punto, operava sull'editor sperimentale. Tutti e tre i programmi giravano sul microelaboratore del Blit. Mentre Pike mi dimostrava la grande utilità di questo sistema a multiprogrammazione, nella parte alta dello schermo apparve un gran numero di icone a forma di granchio, che si affrettarono verso il margine di una finestra e cominciarono a rosicchiarlo. Vedendomi sconcertato, Pike spiegò: «Ah, sì. Ecco i granchi. Per divertimento abbiamo caricato il programma CRABS dal calcolatore ospite.»

Sul calcolatore ospite del Blit stava girando UNIX, uno dei più diffusi sistemi operativi mai sviluppati, creato da Kenneth L. Thompson, ricercatore dei Bell Laboratories che ricevette nel 1983 il premio Turing insieme con Dennis M. Ritchie, suo collega a Murray Hill.

Sotto il mio sguardo affascinato, i granchi (una trentina in tutto), dopo essersi mangiato il margine di una finestra, iniziarono a divorare il testo che vi era contenuto. «Come si fa a fermarli?» «Non puoi» «Sì, ma come si fa a fermarli?»

CRABS, scritto da Luca Cardelli e Mark Manasse nel 1982, è una deliberata violazione delle regole di progettazione del sistema a multiprogrammazione del terminale Blit: il programma di ciascuna finestra deve essere chiuso in se stesso e protetto dai programmi che girano nelle altre finestre. Una volta distribuiti i terminali Blit nei laboratori di Murray Hill, Cardelli e Manasse provarono l'irresistibile tentazione di infrangere queste regole.

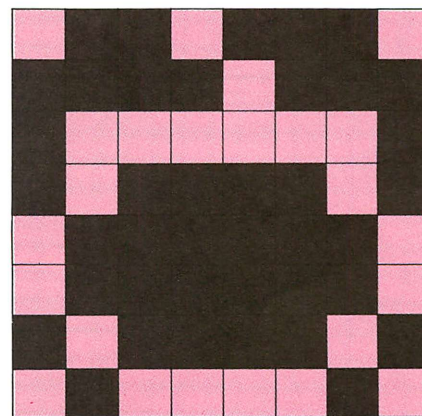
Mentre Pike mi raccontava questa breve storia, i granchi avevano completamente rosicchiato tutte le finestre dello schermo, che ormai sembravano le pagine di un manoscritto medievale appena ritrovato: i testi erano talmente bucherellati che Pike era nell'impossibilità di descrivere quello che stava avvenendo nelle finestre.

Pike spiegò che il terminale originale doveva il suo nome a bitblt, un operatore grafico di basso livello sorprendentemente versatile. «Bitblt» è una contrazione di *bit-boundary block transfer* (trasferimento di blocco a confine di bit), termine che si applica a una procedura che dirige il movimento dell'informazione all'interno della memoria del terminale. Più specificamente, bitblt trasferisce il contenuto di un insieme rettangolare di locazioni di memoria in un altro. Nel corso del processo può compiere semplici operazioni logiche su questo contenuto.

Parte della memoria del terminale Blit è formata da un sottosistema di 100 000 byte dedicato allo schermo di visualizzazione. Un'esplorazione elettronica continua traduce ogni bit di questa memoria in un punto luminoso o scuro sullo schermo, convertendo una matrice unidimensionale di locazioni di memoria in una rappresentazione visiva bidimensionale del suo contenuto. Dato che immagazzinamento e visualizzazione differiscono per numero di dimensioni, le locazioni corrispondenti a un singolo rettangolo dello schermo sono sparpagliate nella memoria sotto forma di blocchi separati di locazioni consecutive; ogni blocco corrisponde a una linea orizzontale del rettangolo. Il complesso di tutte le locazioni di tutti i blocchi costituisce un insieme rettangolare. È compito del bitblt manipolare questi insiemi rettangolari.

Per operare con successo in ambiente di multiprogrammazione, il terminale Blit richiede una notevole quantità di memoria fuori schermo a uso di un programma che stia attualmente girando sullo schermo. Il terminale Teletype DMD 5620, per esempio, riserva 900 000 byte a questo scopo. Anche qui sono essenziali insiemi rettangolari; bitblt è chiamato di frequente a trasferire il contenuto di un insieme della memoria fuori schermo in un insieme rettangolare della memoria di schermo. Per esempio, un certo insieme rettangolare della memoria fuori schermo contiene la lettera *A*. Ogni volta che un programma visualizza *A* come parte della sua uscita testuale, bitblt è chiamato a trasferire questo insieme rettangolare nella memoria di schermo in modo che il carattere possa apparire.

L'analisi di memoria hardware è la caratteristica principale della tecnica della «memoria di transito di quadro» nella progettazione di un terminale grafico. Le memorie di transito di quadro stanno diventando di uso generale, anche nel mercato dei calcolatori personali; basta pensare al MacIntosh.

Nel modo tradizionale di affrontare la grafica, la memoria è consultata più sporadicamente. La visualizzazione di un disegno, per esempio, è risolta attraverso una lista di punti, linee e altri elementi

L'icona base del granchio visualizzata in una griglia

pittorici; quando si deve visualizzare un disegno, viene esplorata la sua lista. Raffigurazioni semplici possono essere esplorate rapidamente perché hanno liste di visualizzazione brevi, mentre visualizzare raffigurazioni complesse può richiedere troppo tempo.

La configurazione di bit che occupa un dato insieme rettangolare della memoria si definisce «mappa di bit». Ogni mappa di bit produce un'immagine specifica sullo schermo, un carattere, un'icona o qualche altro elemento grafico. Il programma CRABS, per esempio, utilizza certe mappe di bit fuori schermo che danno luogo a varie forme dei piccoli crostacei. La seguente successione di 0 e 1 rappresenta un granchio, con gli 1 che indicano il nero e gli 0 il verde:

01101110 ... 11110111 ... 10000001 ... 10111101 ... 01111110 ... 01111110 ... 10111101 ... 01000010

I lettori che vogliano imitare bitblt con carta e penna possono tracciare una griglia quadrata 8 per 8. La successione precedente rappresenterà allora un insieme rettangolare della memoria fuori schermo e la griglia quadrata rappresenterà parte della memoria di schermo. Si esplorino la successione e la griglia, un bit alla volta, e si copi la successione nella griglia. Si riproduca la griglia in una piccola sezione dello schermo annerendo ogni quadrato che contenga un 1; i quadrati che contengono uno 0 sono chiari. Il risultato è l'immagine di un granchio (*si veda l'illustrazione della pagina precedente*).

Ho osservato in precedenza che qualsiasi parte dello schermo che non sia in una finestra ha una tramatura grigia che in realtà è una configurazione di punti. Quando i granchi avevano invaso lo schermo Blit di Pike, avevano vagato per la tramatura grigia finché non avevano incontrato una finestra, che si erano messi a mangiare. Pike aveva così commentato: «Il grigio è la strada per i granchi: non mangiano la loro strada». In realtà, i granchi non mangiano nulla, ricoprono soltanto. Quando CRABS muove una delle sue creature, controlla l'area subito davanti al granchio e, se non è già grigia, la rende tale; poi sposta il granchio nella posizione appena resa grigia. L'operatore bitblt rende possibili entrambe le procedure.

Nella sua forma più semplice, bitblt sostituisce ogni bit d della mappa di bit di destinazione con il corrispondente bit s di una mappa di bit sorgente. Stenograficamente, l'operazione è scritta come una sostituzione:

$$d \leftarrow s$$

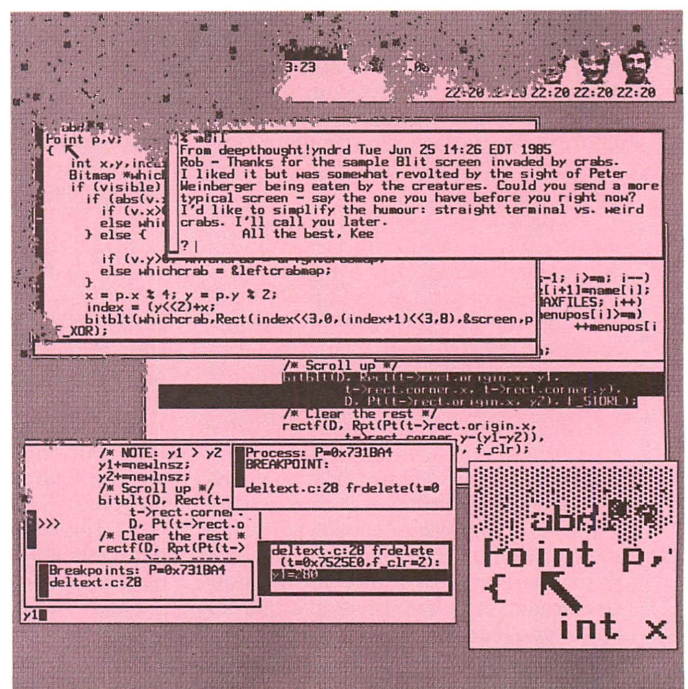
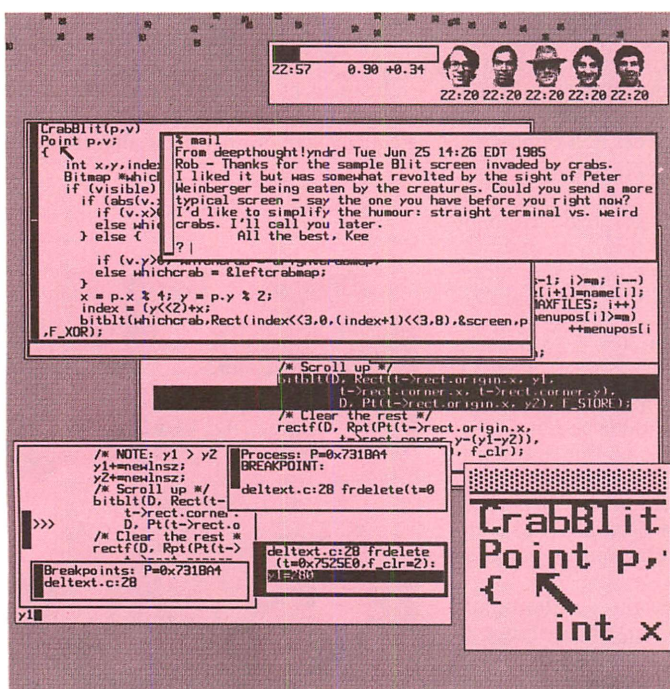
Un'area non grigia viene trasformata in un'area grigia usando bitblt in questa forma. L'area subito davanti al granchio è la mappa di bit di destinazione e una mappa fuori schermo che contiene la tramatura grigia è la mappa sorgente.

Poi CRABS muove il granchio, cancellandolo dalla sua posizione attuale e ridisegnandolo in una nuova posizione. Per compiere queste operazioni, si usa bitblt in una forma più raffinata, riassunta nella notazione:

$$d \leftarrow s \oplus d$$

In questo caso, ogni bit destinazione è sostituito dalla sua combinazione con il corrispondente bit sorgente sotto l'operatore logico detto XOR ($\oplus$): d diventerà 1 se e solo se s o d (ma non entrambi) hanno già questo valore. L'uso dell'operatore XOR è comodo sia per cancellare sia per ridisegnare un granchio. Sfortunatamente, bitblt in questa forma non può usare la figura continua di granchio prodotta dall'esercizio con carta e matita descritto in precedenza.

Consideriamo quello che avviene, per esempio, se la mappa di bit standard di un granchio continuo è sottoposta a XOR sulla tramatura grigia: quando un punto di grigio (un 1) è sottoposto a XOR con una porzione qualsiasi del corpo del granchio (anche un 1), il risultato è un buco verde ($1 \oplus 1 = 0$). Se il corpo ha già un buco in quella posizione, però, il risultato è nero ($0 \oplus 1 = 1$). Questo spiega lo strano aspetto delle otto speciali mappe di bit di granchio effettivamente usate da CRABS per portare i suoi attacchi (*si veda l'illustrazione a pagina 116*). Ogni mappa rappresenta un granchio già perforato da buchi disposti secondo una certa configurazione. Ci sono otto possibili posizioni per un granchio sulla tramatura grigia: le configurazioni riflettono queste posizioni.

CRABS muove un granchio prima cancellandolo e poi ridisegnandolo. Per cancellare un granchio, la versione XOR di bitblt riproduce la mappa di bit del granchio, appropriatamente bucata, sul granchio destinazione. Il granchio desti-

Lo schermo del terminale Blit su cui appare l'uscita di cinque programmi è attaccato dai

nazione scompare; restano soltanto i punti della tramatura grigia che l'operatore XOR si lascia forzatamente dietro. Per disegnare un granchio, bitblt impiega XOR per tracciare un altro granchio adeguatamente perforato sull'area di destinazione. Il granchio appare intatto perché i suoi buchi combaciano perfettamente con la tramatura grigia alla quale deve adattarsi.

L'algoritmo usato dal programma CRABS controlla ogni granchio attraverso cinque passi:

1. Cancella il granchio dalla sua posizione precedente.
2. Determina un nuova posizione.
3. Se la nuova posizione non occupa un luogo grigio, lo rende grigio, ritorna nella posizione precedente e sceglie una nuova velocità casuale.
4. Disegna il granchio nella sua nuova posizione.
5. Modifica a caso la velocità del granchio. Torna al passo 1.

Naturalmente, i granchi si muovono di lato: una nuova posizione, quindi, è sempre su un lato o l'altro di un granchio ed è determinata dall'attuale posizione del granchio e dalla sua velocità. Entrambe queste grandezze sono soggette a cambiamenti essenzialmente casuali nel passo 5. Ci sono molte direzioni possibili; la velocità può variare da uno a sette pixel per mossa.

Durante la mia discussione con Pike, i granchi avevano fatto il loro ingresso dall'alto dello schermo. Si erano poi messi a vagare nelle aree grige spostandosi in generale verso il basso. Arrivati a una finestra, avevano iniziato a rosicchiarla. La loro attività era molto evidente, perché lo sfondo chiaro veniva regolarmente sostituito da piccole chiazze grige. L'effetto era inconfondibile: era come se della carta venisse via via divorata rivelando una sottostante superficie grigia.

Il passo 3 dell'algoritmo di CRABS accresce l'illusione facendo rimbalzare indietro il granchio dall'area appena rosicchiata, come se si fermasse un attimo a digerire prima di ripartire in cerca di altro cibo. La natura casuale del movimento del granchio fa sì che la finestra venga mangiata lungo un fronte piuttosto frastagliato. Senza questa caratteristica, lo schermo assomiglierebbe ancor più al lavoro di scavo di insetti. La casualità dà anche più tempo per reagire all'invasione.

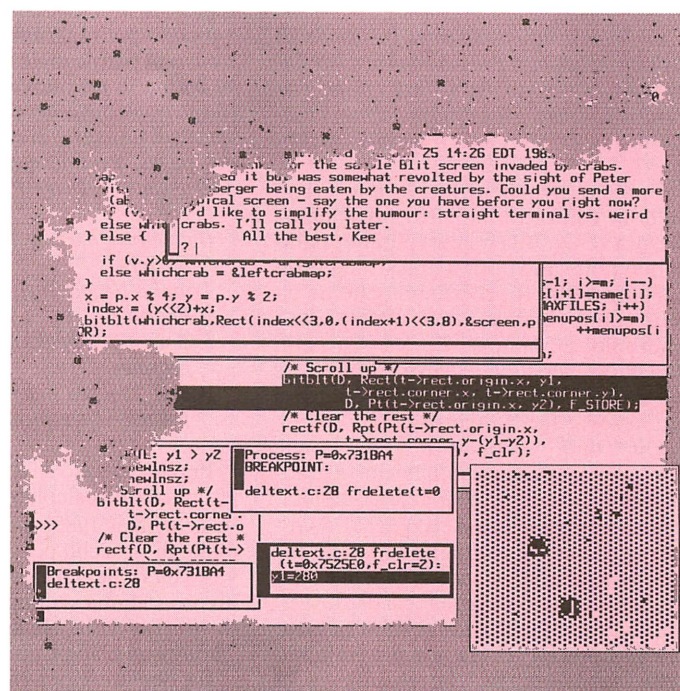
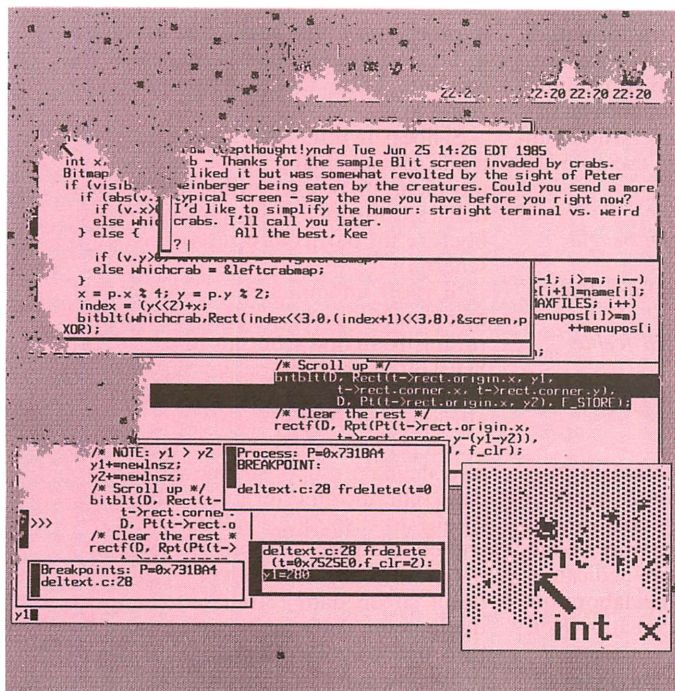
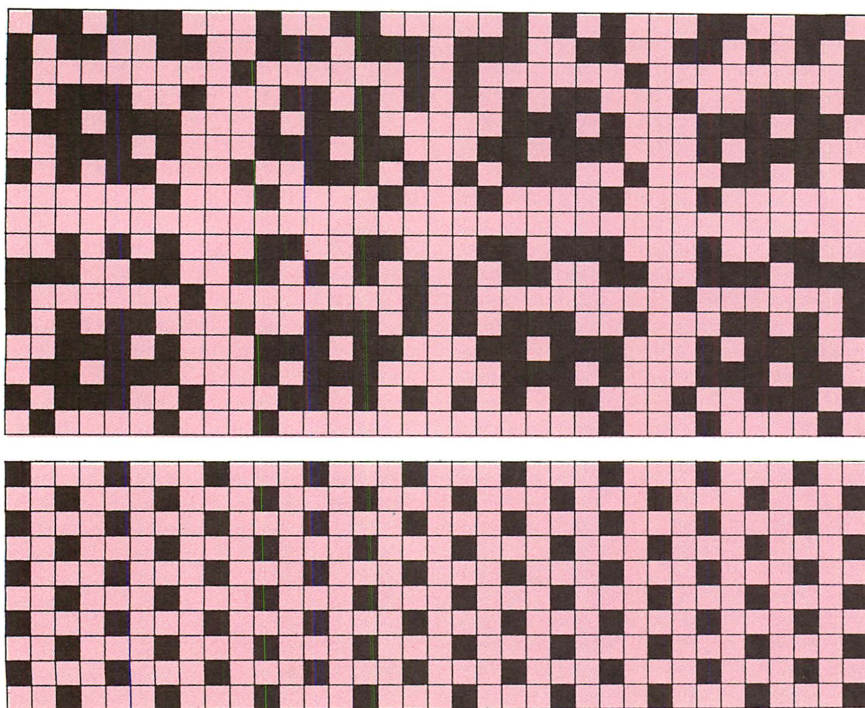
Mentre guardavamo i granchi farsi strada nelle finestre del terminale Blit, Pike mi raccontò le diverse reazioni dei colleghi al loro primo incontro con la moltitudine di piccoli divoratori. Per alcuni l'incontro fu del tutto inatteso: mentre uno di loro era momentaneamente assente, uno degli spiriti più allegri di Murray Hill faceva partire CRABS sul terminale dell'innocente.

Dato che il terminale Blit dispone di un «mouse» che controlla un cursore sullo schermo, una sfortunata vittima potrebbe cercare di usare il congegno per colpire i granchi, ma l'operazione è destinata al fallimento perché i granchi trovano il cursore altrettanto appetibile del testo. Tra un morso e l'altro, il cursore si rigenera: i granchi potrebbero quindi essere rallentati in questo modo, ma non fermati e le finestre continuerebbero a disintegrarsi.

La vittima potrebbe cercare di coprire i granchi con una finestra. Anche questo li rallenterebbe, ma un granchio intrappolato sotto una finestra si rosicchierebbe intorno un'area sempre più larga e infine si libererebbe.

Chiesi nuovamente a Pike come si potevano fermare i granchi e questa volta si lasciò commuovere: «Spegni il terminale e fallo ripartire». La manovra funziona, ma significa ricominciare daccapo la procedura di ingresso nel calcolatore ospite.

Cardelli e Manasse, gli autori di CRABS, raccontano in un memorandum informale la genesi del programma. Affascinati dalle regole di progettazione del terminale Blit, presero in esame le possibili violazioni. Consultarono anche Pike, che diede loro i necessari frammenti di software. I due programmatori trovarono che alcune delle possibili violazioni delle regole di progettazione potevano essere di reale utilità. Un esempio è il programma LENS (lente) che ingrandisce una porzione dello schermo. (Nelle figure qui sotto si può vedere la finestra LENS che ingrandisce un testo infestato dai granchi.) LENS è una violazione perché un processo che avviene entro una finestra non dovrebbe avere accesso diretto al contenuto di un'altra finestra.

In origine, Cardelli e Manasse avevano in mente di emulare un videogioco chiamato QIX, invece inventarono un programma detto MEASLES (morbillo), in cui dei pallini fluttuavano per lo

granchi. Un utente può far fallire un attacco del genere spegnendo il terminale e ricollegandolo

Mappe di bit dei granchi perforate danno otto possibili posizioni sulla tramatura grigia

schermo nelle aree grige tra le finestre. Prove iniziali dimostrarono rapidamente che i pallini potevano essere controllati collocandovi sopra una finestra.

Chiaramente, gli autori simpatizzavano con le loro creature: «Improvvisamente, quei poveri pallini non sanno più dove andare, non hanno più aree grige dove correre. Sono raggelati, paralizzati di terrore e sepolti sotto una finestra».

La loro prima soluzione fu quella di far lampeggiare ogni pallina intrappolata, nel tentativo di disturbare l'imprigionatore. Questo programma venne chiamato ANGRY MEASLES (morbillo arrabbiato).

I pallini arrabbiati, però, non erano nulla in confronto all'altro programma che ora veniva alla mente: perché i pallini non potevano aprirsi una strada fuori dalle finestre? Il programma che ne risultò venne chiamato HUNGRY MEASLES (morbillo affamato). Scrive Cardelli: «La nuova versione... aveva un carattere del tutto diverso. Non era più qualcosa di grazioso: faceva paura».

Qualcuno, vedendo il programma, disse che i pallini assomigliavano a granchiolini. Cardelli e Manasse completarono la somiglianza, ed ecco nato CRABS. Altri programmi, tra cui quello che lascia sullo schermo tracce di uccelli e altri animali, vennero poi sulla scia di CRABS.

Le vittime degli scherzi sono passate al contrattacco. C'era un programma, detto SQUISHCRABS, che esaminava la memoria del Blit alla ricerca del programma CRABS. Se lo trovava, SQUISHCRABS uccideva il programma e trasformava in bolle tutti i granchi che trovava. Sfortunatamente, una modificazione del software del terminale ha reso inefficace SQUISHCRABS.

Molti altri sono i programmi creati per svago a Murray Hill; tra gli altri mi è stato presentato AUTOPUNNER [*to pun* = far giochi di parole], scritto da Ron Hardin. Questo programma traduce un testo inglese in una successione di fonemi, che vengono poi raggruppati, trasformati di nuovo in parole inglesi e visualizzati sotto forma di un nuovo testo. Il risultato (dopo qualche intervento umano) ha l'aspetto di una serie di strani versi liberi un po' pazzi. Prendiamo, per esempio, la distorsione di *Peter Piper:*

Better buy perfect Topeka beagle buffers
Topeka beagle buffers sputter fiber beaks
Effeter fiber beaks abetted feeble puppies
Worst Topeka beagle puppies feature viper-pique.

Più finalizzato è il tentativo di programmare un calcolatore perché giochi a scacchi. Nel 1980, il campionato mondiale di scacchi per calcolatori venne vinto da Belle, un calcolatore dedicato e un programma sviluppato nei laboratori di Murray Hill. Ne sono creatori Ken Thompson e Joe Condon.

Il calcolatore per gli scacchi, dedicato alla ricreazione, riporta la discussione alla tensione creativa tra lavoro e gioco. Alan J. Perlis, professore di scienze del calcolatore alla Yale University, esprime molto efficacemente l'atteggiamento di grande apertura che si incontra tra gli studiosi dei calcolatori in un libro intitolato *Structure and Interpretation of Computer Programs*, scritto da Harold Abelson, Gerald Jay Sussman e Julie Sussman.

«Penso che sia straordinariamente importante che noi, che ci occupiamo di scienza dei calcolatori, ci divertiamo. All'inizio era un gran divertimento.» Ma, secondo Perlis, il divertimento è stato offuscato quando agli studiosi di calcolatori si è cominciato a chiedere di dedicarsi con eccessiva attenzione ai dettagli pratici dei loro sogni e dei loro progetti. Questa preoccupazione riguarda più propriamente chi pilota l'evoluzione della scoperta verso congegni produttivi. «Abbiamo cominciato a sentirci come se fossimo davvero responsabili del successo e della perfezione, della assenza di errori nell'uso di quelle macchine. Non credo che le cose stiano così; penso che la nostra responsabilità stia nello spingerle ai limiti, nel ricercare nuove direzioni per la loro evoluzione e nel continuare a trarne motivo di divertimento.»

Il divertimento e la produttività della scienza dei calcolatori stanno proprio nel senso dello sviluppo, nel dispiegarsi di idee che sembrano venire metà dalle persone e metà dalle macchine stesse. Si può arrivare a dire che, senza divertimento, il vero progresso sia ben scarso.

Non posso terminare questo articolo senza tornare al terminale Blit e all'operatore bitblt. Pike ha proposto due interessanti rompicapo che dimostrano la potenza e la flessibilità di bitblt. I lettori dovrebbero immaginare di avere in dotazione le due versioni di bitblt ricordate in precedenza: una sostituisce la mappa di bit destinazione con la mappa di bit sorgente; l'altra invece effettua l'operazione XOR.

Si liberi un rettangolo. Senza ricorrere a una speciale mappa di bit fuori schermo piena di 1, si usi bitblt nella forma della sostituzione o in quella XOR per liberare un particolare rettangolo sullo schermo.

Si ruoti un'immagine. Un'immagine quadrata $n \times n$ occupa l'angolo superiore sinistro di uno schermo Blit $2n \times 2n$. Si usi bitblt per ruotare l'immagine di 90 gradi. Si supponga che lo schermo sia altrimenti vuoto.

Nel primo problema non è consentito fare alcuna presupposizione sull'aspetto dello schermo all'esterno del rettangolo. In questo caso, quale mappa di bit potrebbe svolgere il ruolo di sorgente?

In entrambi i problemi l'effetto di bitblt su un dato rettangolo (indipendentemente dalla sua forma) è considerato una singola operazione. Il primo problema può essere risolto con un unico bitblt e il secondo con $3n + 1$. Qualcuno sa trovare un modo più veloce?

Tappezzeria per la mente: immagini al calcolatore quasi, ma non del tutto, ripetitive

di A. K. Dewdney

Le Scienze, novembre 1986

La comune carta da parati è stampata da un cilindro rotante su cui è inciso un disegno. Ruotando, il cilindro continua a stampare sempre lo stesso disegno. Solo un calcolatore, però, può riprodurre certe complicate raffigurazioni che io chiamo «tappezzeria per la mente». Queste raffigurazioni non si ripetono, almeno non esattamente; ognuna continua a manifestarsi in nuovi contesti e configurazioni. Che cosa cambia e che cosa si conserva nel passaggio da un'apparizione alla successiva?

Gli esempi a colori contenuti nel mio attuale campionario rappresentano il frutto di tre tecniche molto differenti tra loro. I programmi che generano quelle immagini, e che hanno un grado di difficoltà variabile dall'estremamente semplice al facile, mi sono stati forniti da tre lettori: John E. Connett dell'Università del Minnesota, Barry Martin di Birmingham, Inghilterra, e Tony D. Smith di Essendon, Australia.

Il programma di Connett si fonda sul cerchio, ma esalta le varietà di disegno basate sul quadrato. L'apparente enigma mi induce a chiamarlo CIRCLE^2. In poche parole, il programma applica la formula analitica di un cerchio, $x^2 + y^2$, per assegnare un colore al punto di coordinate x e y. Darò più avanti i dettagli. Nel frattempo potete scoprire con sorpresa, come è successo a me, che questa tappezzeria contiene molto più che un insieme di cerchi concentrici; in particolare, se vi allontanate possono emergere intricate configurazioni di graziosi quadrati (*si vedano le illustrazioni della pagina successiva*). C'è qualcosa di misterioso in tutto questo.

Forse non vi sorprenderà il fatto che CIRCLE^2 sia stato ispirato dall'insieme scoperto da Benoit B. Mandelbrot, del Thomas J. Watson Research Center dell'IBM; la ressa di forme e colori che circonda l'insieme di Mandelbrot si basa su una singola funzione matematica applicata ripetutamente al suo stesso risultato per ogni numero complesso in una regione del piano. Ogniqualvolta il valore iterato della funzione raggiunge il valore assoluto 2, il numero di iterazioni necessarie per raggiungere quel valore determina il colore del punto corrispondente.

Connett, che non aveva a disposizione un video a colori, assegnava il nero ai punti che raggiungevano 2 in un numero pari di iterazioni e il bianco a quelli che raggiungevano 2 in un numero di iterazioni dispari. Ne risultavano immagini dell'insieme di Mandelbrot più che accettabili, ma Connett si sentì stimolato a esplorare altre formule. Scelse la formula $x^2 + y^2$ e contemporaneamente abbandonò l'iterazione. Il suo programma esamina in modo sistematico una sezione a griglia del piano; in ogni punto (x,y) calcola la formula e tronca il valore risultante alla parte intera. Se questa è un numero pari, colora il punto (x,y) di nero; se è dispari, il punto viene colorato di bianco (lasciato vuoto).

Temo di aver perso metà del mio pubblico, che ha già capito il programma ed è corsa al più vicino calcolatore per scriverlo. È talmente semplice! In notazione algoritmica, CIRCLE^2 è formato da una sezione di ingresso seguita da un doppio ciclo:

```
inserisci anga, angb
inserisci lato

per i ← 1 a 100
    per j ← 1 a 100
        x ← anga + (lato × i/100)
        y ← angb + (lato × j/100)
        z ← x² + y²
        c ← int (z)
        se c è pari, allora scrivi (i, j)
```

Per prima cosa il programma fa chiedere al calcolatore le coordinate (*anga*, *angb*) dell'angolo a sinistra in basso del quadrato da esaminare. La variabile *lato* è la lunghezza del lato del quadrato da esaminare. Per esempio, se l'utente batte alla tastiera -15 e -20 per le coordinate d'angolo e 87 per il lato, il programma passa in rassegna una matrice di 100×100 punti in una regione quadrata del piano avente per lato 87 unità e il cui angolo a sinistra in basso è il punto $(-15, -20)$. Nell'abbozzo che ho dato del programma ho assunto che i limiti delle iterazioni vadano da 1 a 100, ma possono essere regolati per adattarsi ai margini della periferica di uscita che si usa.

Il doppio ciclo procede lungo la griglia quadrata e per ogni coppia di indici (i, j) calcola le coordinate del punto (x, y) a cui la coppia corrisponde. Il ciclo eleva poi al quadrato x e y, assegna la somma dei due quadrati a z e la tronca alla parte intera. Il più grande intero minore o uguale alla somma è immagazzinato come variabile c. Se c è divisibile per z, il punto (x, y) viene visualizzato, come pixel colorato su un monitor o come puntino nero se si usa una stampante. Se c è dispari, non viene visualizzato alcun punto.

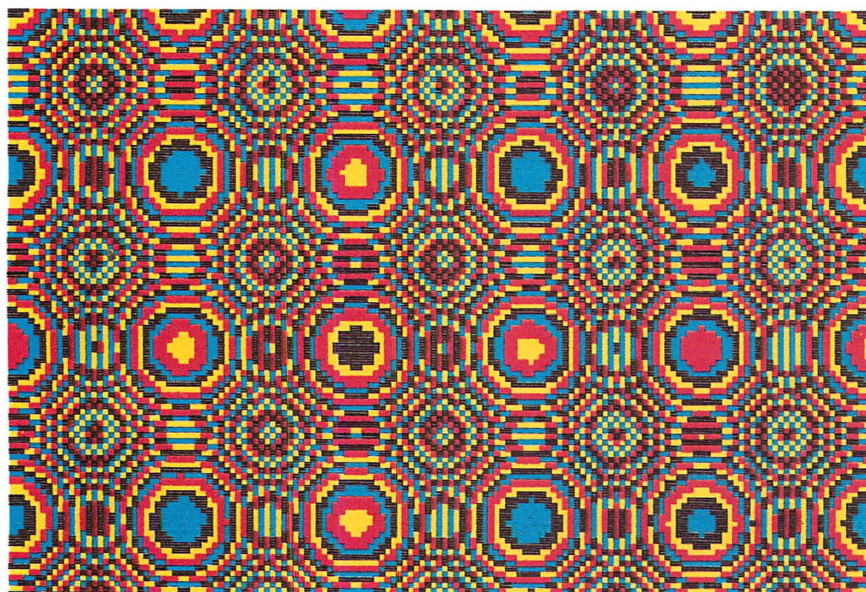
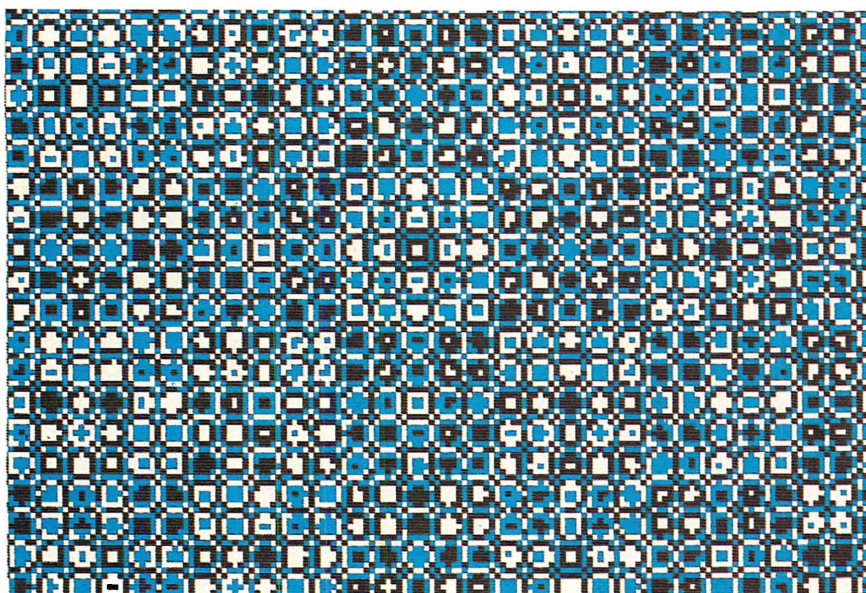
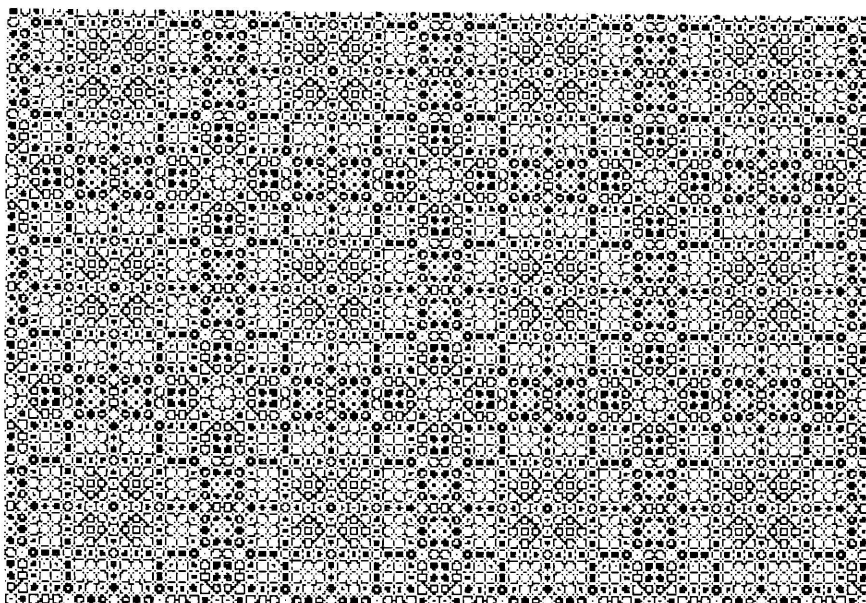
I lettori che vogliano (ri)creare la tappezzeria di Connett non devono preoccuparsi troppo della assegnazione dei colori giusti: la maggior parte delle configurazioni è ugualmente straordinaria a colori invertiti. In realtà si possono usare anche più di due colori: invece di stabilire se c è pari o dispari, lo si divida per il numero di colori voluto e si assegnino i diversi resti ai diversi colori. Per esempio, le illustrazioni della pagina successiva sono state generate scegliendo, rispettivamente, due, tre e quattro colori.

Più piccolo è il quadrato in esame, più vicino appare il piano all'osservatore e maggiore è l'ingrandimento dell'immagine di CIRCLE^2. A differenza, però, del procedimento per colorare i dintorni dell'insieme di Mandelbrot, il programma di Connett non porta a un regresso infinito di configurazioni sempre più piccole. Con forti ingrandimenti intorno all'origine (0,0) appare un insieme di cerchi concentrici. Con ingrandimenti maggiori c'è un grande disco nero al centro dello schermo: la parte intera di ogni punto del disco è pari a zero. Poi tutto lo schermo è nero.

Si può apprezzare meglio la bellezza della tappezzeria di Connett riducendo l'ingrandimento: come se ci si allontanasse dalla parete. I cerchi concentrici si dissolvono in un'interessante disposizione di cerchi primari e secondari simili a disegni moiré. A ogni riduzione dell'ingrandimento compaiono come per magia nuovi disegni apparentemente diversi. C'è in agguato un regresso all'infinito? Si direbbe una domanda imbarazzante, ma confido che i lettori saranno in grado di far luce sulla questione.

Anche a Barry Martin, della Aston University di Birmingham, l'ispirazione è venuta dall'insieme di Mandelbrot. Martin ha adottato l'idea di Mandelbrot di iterare una formula a partire da un seme numerico, ma le somiglianze si fermano qui. Mentre le configurazioni di Mandelbrot emergono da numeri complessi, la tappezzeria di Martin si basa su iterazioni di comuni numeri reali. Inoltre, i semi numerici delle configurazioni di Mandelbrot sono gli infiniti punti di una regione del piano, mentre le configurazioni del programma di Martin crescono da un unico seme.

Martin suggerisce di mettere alla prova la seguente coppia di formule che generano immagini multicolori stupende e ricche di dettagli (*si vedano le illustrazioni delle pagine 119, 120, 121*).

$x \leftarrow y - \text{segn}(x) \times [\text{ass}(b \times x - c)]^{1/2}$
$y \leftarrow a - x$

Cerchi e quadrati modulo 2, 3 e 4 dal programma CIRCLE² *di John E. Connett*

Qui la funzione segn(x) assume il valore 1 se x è positivo e -1 se x è negativo; la funzione ass($b \times x - c$) è il valore assoluto dell'espressione $b \times x - c$. Le configurazioni possono essere molto diverse tra loro a seconda dei valori assegnati alle lettere a, b e c, che nella formula sono costanti numeriche.

Le formule sono scritte in una sorta di stenografia matematica: si deve intendere che per x e y si usa un insieme di valori a destra delle frecce e da questi si calcola a sinistra un nuovo insieme di valori. I nuovi valori di x e y sostituiscono poi i vecchi valori a destra delle frecce e si ripete il calcolo. Il programma, che io chiamo HOPALONG, salta così da un punto a un altro. Inizia nel punto per cui x e y sono entrambi uguali a 0, cioè nell'origine, mentre il punto successivo potrebbe essere a destra in alto e quello ancora successivo a sinistra in basso.

Un calcolatore traccia i punti così rapidamente da dare l'impressione che una minuta pioggia elettronica stia cadendo sullo schermo: centinaia e poi migliaia di punti gocciolano sul monitor e presto comincia a emergere una configurazione. Per esempio, se si pone a uguale a -200, b a 0,1 e c a -80, si forma una configurazione grosso modo ottagonale (*si veda l'illustrazione in basso della pagina a fronte*). Se si ingrandisce il disegno e si colora ogni punto a seconda del numero di salti necessario per raggiungerlo, il disegno diventa un meraviglioso cartiglio (*si veda l'illustrazione in alto della pagina a fronte*). Con altri valori di a, b e c appaiono nuovi disegni: provate con a uguale a 0,1, b uguale a 1 e c uguale a 0 (*si veda l'illustrazione in alto di pagina 120*), oppure con a pari a $-3,14$, b a 0,3 e c a 0,3 (*si veda l'illustrazione in basso di pagina 120*).

L'algoritmo per HOPALONG è quasi altrettanto facile di quello per CIRCLE ²:

inserisci *num*
inserisci a, b, c

$x \leftarrow 0$
$y \leftarrow 0$
per $i \leftarrow 1$ a *num*
 disegna (x,y)
 $xx \leftarrow y - \text{segn}(x) \times [\text{ass}(b \times x - c)]$
 $yy \leftarrow a - x$
 $x \leftarrow xx$
 $y \leftarrow yy$

Giunti a questo punto, ecco un altro fuggi fuggi di lettori che corrono a scriversi il programma. La ricompensa per voi che siete rimasti qui è una spiegazione più dettagliata del programma di Martin e una descrizione del terzo tipo di programma per tappezzeria.

Per far girare HOPALONG si inserisce il numero totale di iterazioni come variabile *num*; inoltre si inseriscono i valori di a, b e c. Più grande è il valore di *num* più dettagliato è il disegno. Per esempio, se *num* è 10 000, il programma disegnerà 10 000 punti sullo schermo, ma per alcu-

ni valori di *a*, *b* e *c* questo è solo l'inizio. Se *a* è uguale a −1000, *b* a 0,1 e *c* a −10, il disegno a basso ingrandimento sembra la scorza di un limone quadrilobato (*si veda la parte inferiore dell'illustrazione a pagina 121*). Se si prolunga il programma da 10 000 punti a 100 000 e poi a 600 000, la filigrana diventa sempre più elaborata (*si veda la parte superiore dell'illustrazione a pagina 121*).

L'algoritmo può funzionare così com'è, ma può anche essere migliorato; si potrebbe aggiungere, per esempio, la possibilità di spostare punti che si trovano fuori dello schermo o di comprimere nella regione visibile regioni che si trovano fuori dello schermo. Se si aggiungono queste caratteristiche, all'inizio del programma si devono specificare altri tre parametri per determinare la posizione e la scala. Si deve poi modificare il corpo del ciclo principale: subito dopo il calcolo di *x* e *y*, la versione migliorata di HOPALONG somma a *x* e *y* i cambiamenti di posizione e moltiplica il risultato per il fattore di scala.

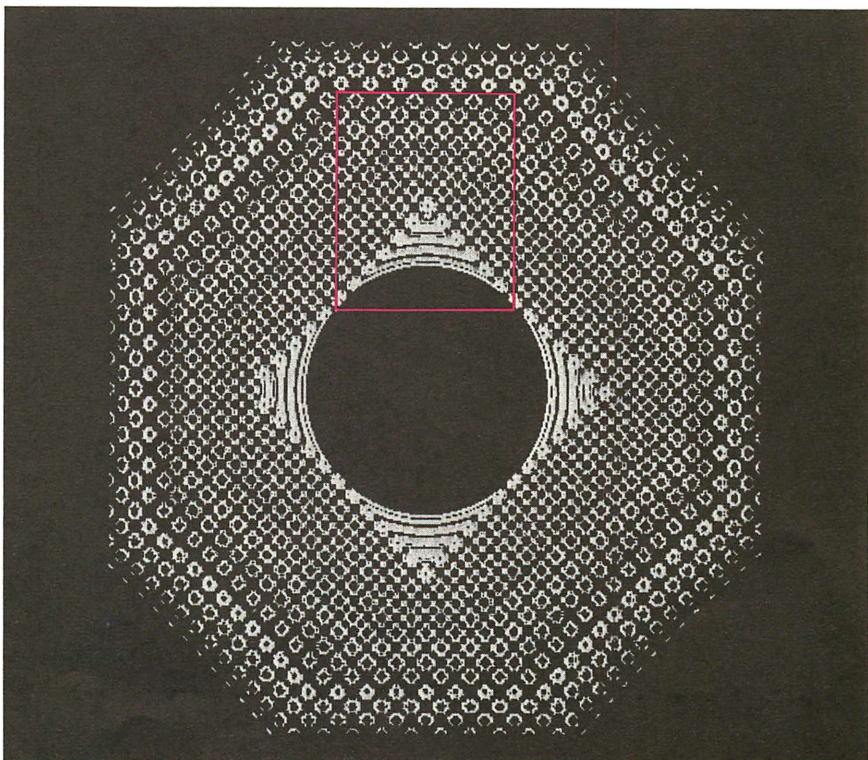
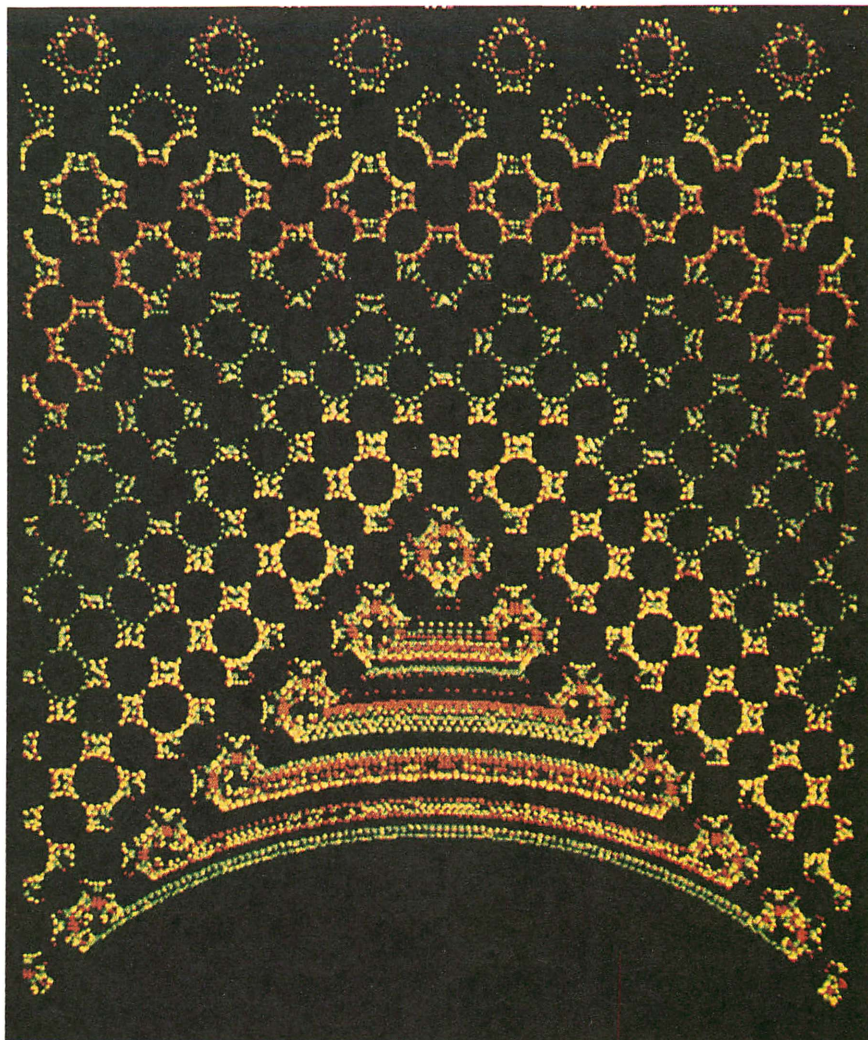
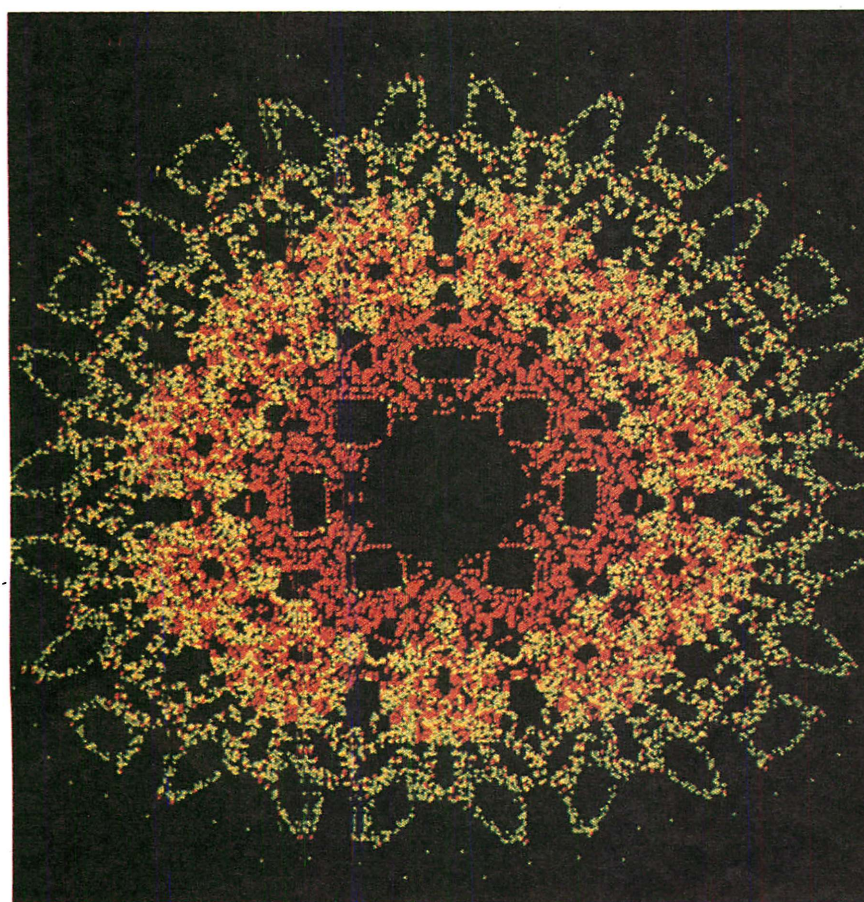
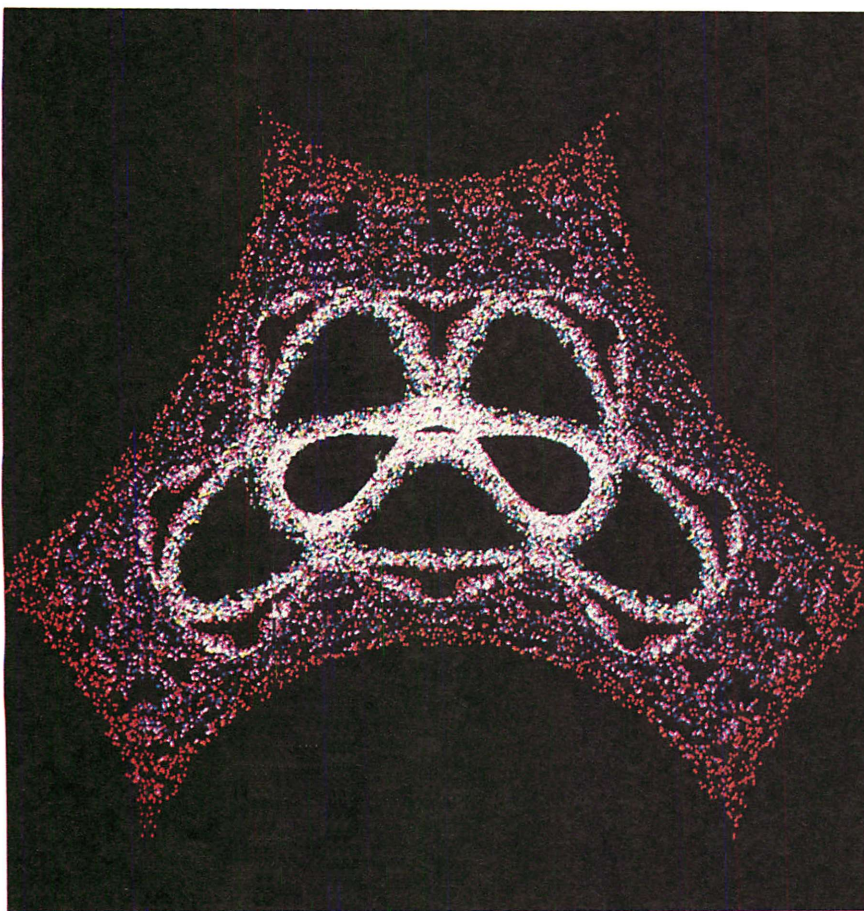
Martin non ha perso di vista l'analogia con la carta da parati: «Credo che assisteremo tra breve a un'esplosione di procedimenti per generare disegni con grandi conseguenze commerciali; ci si può aspettare, per esempio, di vedere nei prossimi anni tappezzerie e tessuti da "designer". Le raffigurazioni saranno prodotte dal cliente scegliendo semplicemente qualche numero». Altrettanto ottimismo Martin dimostra per le conseguenze in biologia matematica. Guardiamo ancora il limone quadrilobato. Gli ingrandimenti presentano dettagli che ricordano fortemente fasci vascolari: forse è la scorza esterna di una monocotiledone in sezione trasversale?

Forse i lettori gradiranno esplorare le configurazioni generate da un'altra coppia di formule di iterazione, anch'esse proposte da Martin:

$$x \leftarrow y - \mathrm{segn}(x)$$
$$y \leftarrow a - x$$

In queste formule si deve specificare solo la variabile *a*. Martin ha scoperto una serie interessante di disegni con *a* molto vicino a pi greco (con una differenza, rispetto a pi greco, non superiore a 0,07).

Il terzo tipo di tappezzeria mentale va riservato a stanze in cui ci si dedichi a intense meditazioni. I disegni vanno dalle complessità persiane agli arzigogoli incaici (*si vedano le illustrazioni delle pagine 122 e 123*) e i metodi per ottenerli sono del tutto diversi dalle tecniche descritte in precedenza. Tony D. Smith, della PICA Pty. Ltd. in Australia, ha progettato complesse variazioni dell'automa cellulare ad autoriproduzione inventato nel 1960 da Edward Fredkin del Massachusetts Institute of Technology (*si veda l'articolo a pagina 66*).

Che cos'è l'automa cellulare di Fredkin? Immaginate una schiera bidimensionale infinita di cellule quadrate. In

Tappezzeria «frattale» dal programma HOPALONG *di Barry Martin*

Mandala sul numero 7 generati dalla formula di iterazione di Martin

ogni istante, ciascuna cellula si trova in uno di due stati possibili: viva o morta. Da qualche parte, un immaginario orologio segna il tempo. Il destino di ciascuna cellula è determinato dalle quattro cellule adiacenti: se, per un battito dell'orologio, il numero di adiacenti vive è pari, la cellula sarà morta al battito successivo, indipendentemente dal suo stato precedente. Se il numero di adiacenti vive è dispari, la cellula sarà viva al battito successivo. La stessa regola è applicata simultaneamente a tutte le cellule.

L'automa di Fredkin è strettamente collegato al gioco Vita, inventato da John Horton Conway. L'automa di Fredkin, però, venne inventato prima di quello di Conway ed è molto più semplice. Inoltre, ha una sorprendente proprietà che Vita non possiede: qualsiasi configurazione iniziale di cellule vive si sviluppa, dopo una serie di generazioni (battiti dell'orologio), in quattro copie di se stessa. Dopo molte altre generazioni le copie diventano 16, poi 64 e così via. La tappezzeria più bella compare durante le generazioni intermedie.

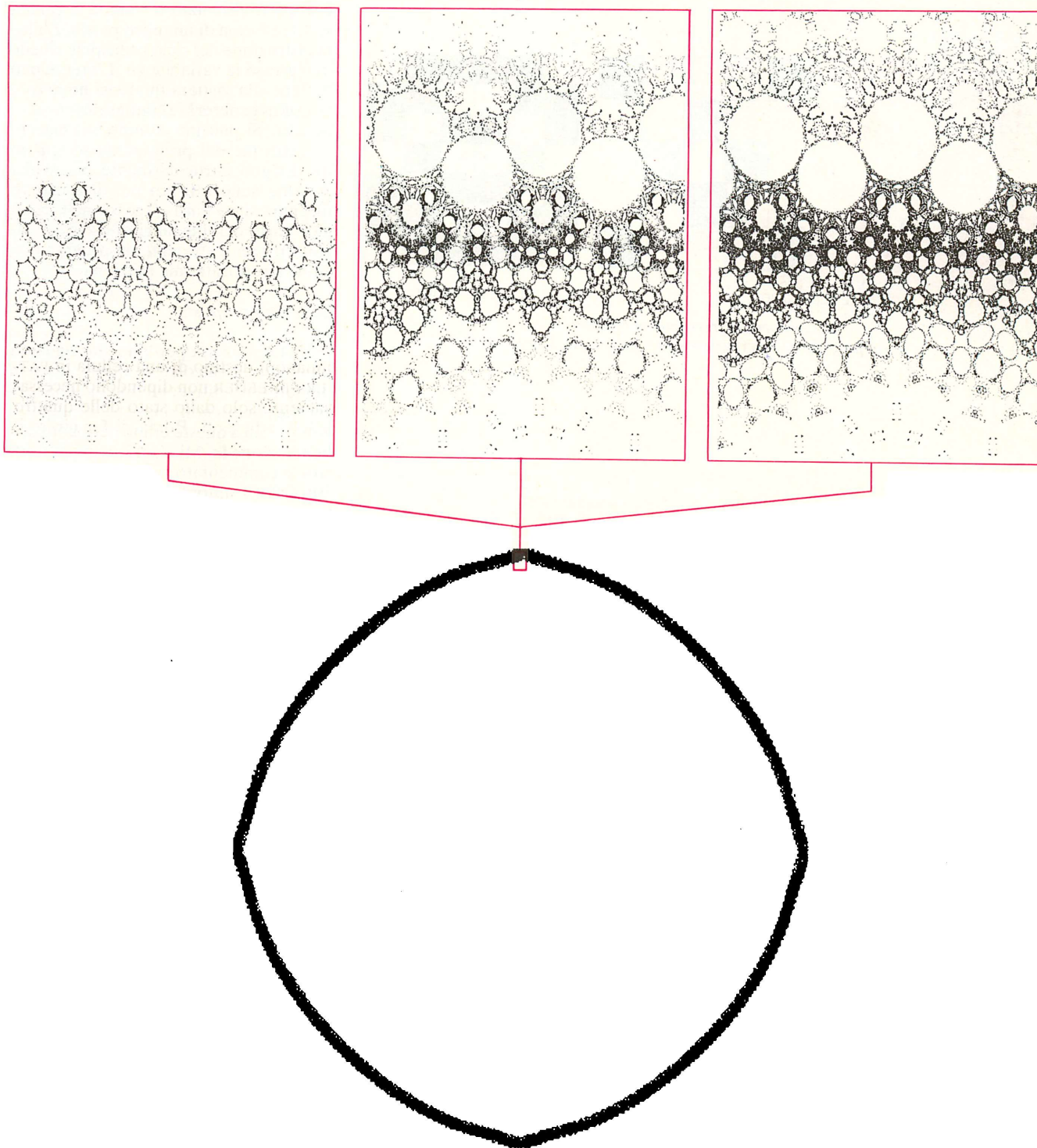
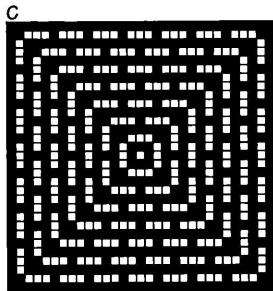
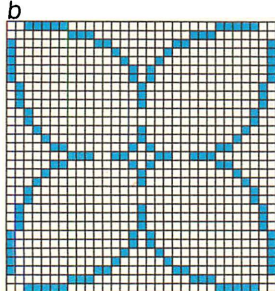
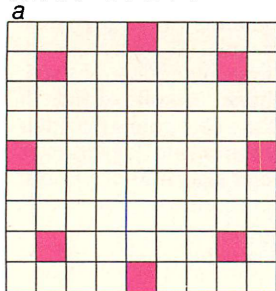
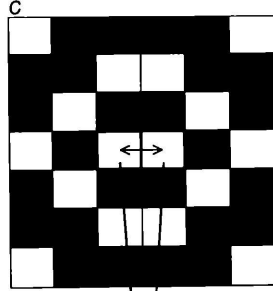
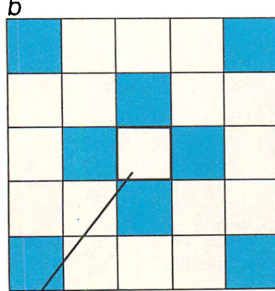
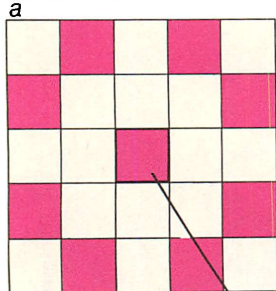
Il programma di Smith per stampare una tappezzeria di Fredkin generalizzata è molto versatile e prende il nome di PATTERN BREEDER. Le regole che determinano il destino di una cellula in PATTERN BREEDER non dipendono necessariamente solo dallo stato delle quattro cellule adiacenti. Prima di far girare il programma si deve specificare la configurazione di cellule circostanti che costituirà l'intorno attivo di ciascuna cellula. Il programma applica poi la stessa regola pari-dispari scelta per l'automa originale di Fredkin. A ogni battito dell'orologio, se il numero di cellule vive dell'intorno attivo è pari, la cellula bersaglio sarà morta nella generazione successiva. In caso contrario, la cellula sarà viva.

PATTERN BREEDER lavora su qualsiasi configurazione di cellule fornita dall'utente. Per esempio, la configurazione iniziale indicata con *a* nell'illustrazione in alto di pagina 122 dà luogo alla parte in rosso del disegno che appare nell'illustrazione sottostante: per ciascuna cellula bersaglio e per ogni stadio dell'evoluzione del disegno, l'intorno attivo è lo stesso. È a sua volta un disegno complesso che comprende tutte le cellule in colore di una matrice 5 per 5, indicato ancora con *a* nell'illustrazione in alto di pagina 122. Si noti che l'intorno attivo include in questo caso la cellula bersaglio stessa. Per applicare la regola, si conti il numero di cellule vive che coincidono con le cellule dell'intorno attivo; se la cellula bersaglio è attualmente viva, viene inclusa nel conto. Le configurazioni iniziali e l'intorno attivo associato a ciascuna di esse sono mostrati per altre due immagini. Quelle contrassegnate con *b* corrispondono alla parte azzurra del disegno in basso a destra e quelle contrassegnate con *c* corrispondono all'illustrazione di pagina 123. C'è una complicazione ulteriore per l'intorno attivo *c*: la

stessa cellula bersaglio oscilla in generazioni successive da sinistra a destra e poi di nuovo indietro al centro dell'intorno.

Non starò a delineare PATTERN BREEDER in tutta la sua raffinatezza, ma descriverò un programma più semplice, che chiamo FREDKIN, trasformabile in uno più generale che presenti alcune caratteristiche di PATTERN BREEDER.

```
inserisci la configurazione iniziale
S  per ogni cellula della matrice
     cont ← 0
     per ciascun adiacente della cellula
       se l'adiacente è vivo
         allora cont ← cont + 1
     se cont è pari
       allora cell ← 0
       altrimenti cell ← 1
             disegna cell
   inserisci go
   vai a S
```

Una delle fortune dello scrivere algoritmi è la disponibilità di tanti livelli di descrizione. La linea di demarcazione tra una descrizione generale e una irresponsabile vaghezza è molto sottile. Si noti che FREDKIN, così come l'ho delineato, occupa uno strato leggermente più rarefatto degli algoritmi descritti in precedenza. Per esempio, l'istruzione «inserisci la configurazione iniziale» richiederà molte istruzioni in un qualsiasi

Modello di fusto vegetale e suoi fasci vascolari generati da HOPALONG

Regole per generare automi cellulari nel programma* PATTERN BREEDER *di Tony D. Smith

Due immagini generate da* PATTERN BREEDER *e sovrapposte

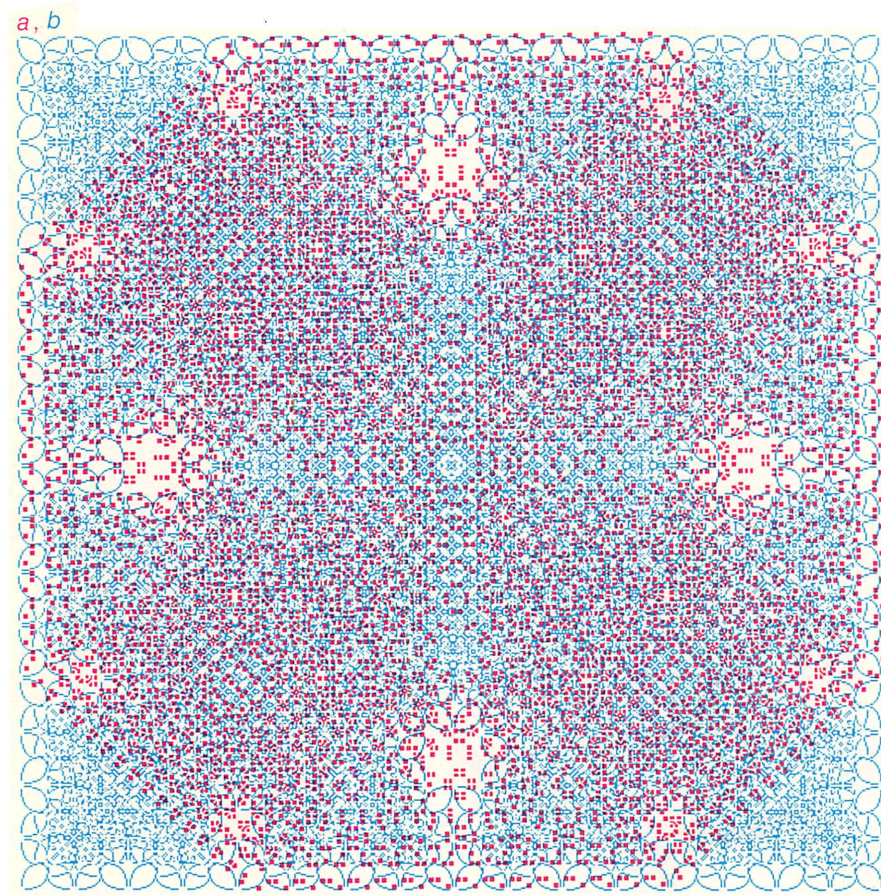
linguaggio di programmazione. Comunque siano, queste istruzioni comprenderanno un doppio ciclo, con due indici *i* e *j*. Un altro doppio ciclo è nascosto nell'istruzione «per ciascuna cellula della matrice». Qui i due indici forniscono le coordinate dei punti dello schermo di visualizzazione o della stampante.

Nel ciclo principale del programma, FREDKIN mette semplicemente in atto la regola per l'evoluzione di un disegno dato. Conta il numero di adiacenti vive per ciascuna cellula (*i*, *j*); poi, se il numero è dispari, visualizza o stampa la cellula sotto la forma di un unico punto. L'ultima istruzione del ciclo principale chiede in ingresso la variabile *go*. L'utente può battere alla tastiera qualsiasi numero e FREDKIN genererà la configurazione successiva. Si può interrompere a piacere l'esecuzione del programma se appare un disegno particolarmente piacevole. A volte sono utili un po' di astuzia e un enunciato «Vai a» («go to» nella maggior parte dei linguaggi di programmazione): non è strategia da programmazione strutturata, ma funziona.

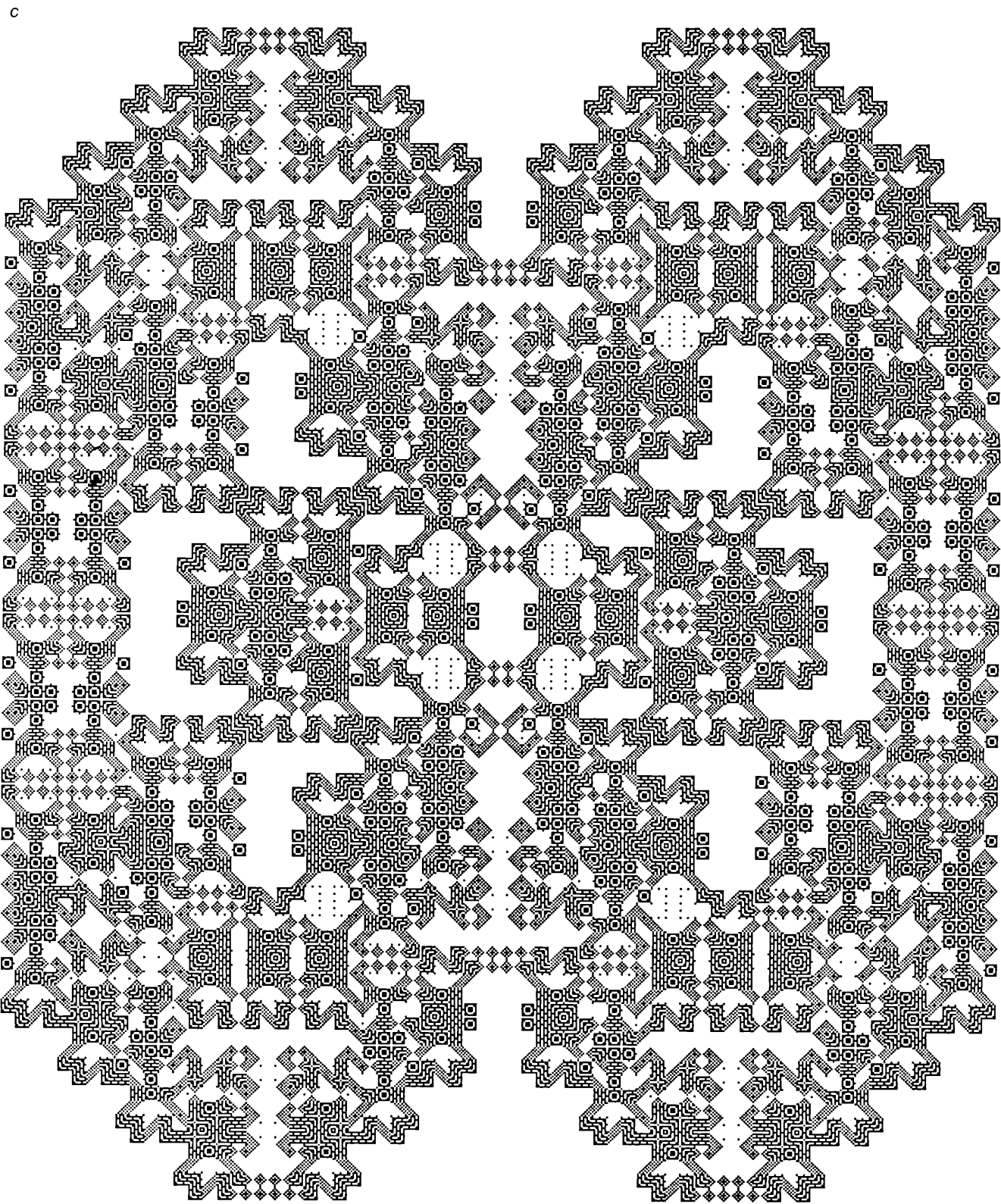
Soluzioni proposte

La tappezzeria per la mente riguarda immagini al calcolatore quasi, ma non del tutto, ripetitive: un tipo di tappezzeria che ancora non si è visto. Configurazioni simili a queste erano però note una generazione fa. Michael Rossman, scrittore e commentatore politico che vive a Berkeley, California, coniò la frase «tappezzeria per la mente» nel 1971. Si riferiva ai delicati disegni che si vedono a occhi chiusi dopo l'assunzione di LSD.

Con il programma di Connett, la comparsa di una tappezzeria (ripetizioni orizzontali e verticali) può essere spiegata in parte come fenomeno moiré: sono implicite due configurazioni e la loro sovrapposizione crea l'effetto. La prima configurazione è la griglia rettangolare di pixel che costituisce lo schermo di visualizzazione. La seconda configurazione è una serie di anelli concentrici che rappresentano i punti del piano per i quali il procedimento di Connett genera un numero dispari. La tappezzeria nasce come risultato di figure di interferenza ripetitive nelle direzioni orizzontale e verticale. Può capitare che un gran numero di punti consecutivi della griglia cada all'interno degli anelli; il gruppo successivo cadrà quindi all'esterno, e così via. A mano a mano che la distanza dall'origine aumenta, gli anelli diventano sempre più piccoli, garantendo che i centri e i colpi mancati avvengano lungo qualsiasi linea di punti della griglia.

Un sostegno a queste affermazioni viene dall'esperienza di Paul Braun di Simi Valley, California, che si è fatto vincere dall'impazienza. Lo schermo del calcolatore ci metteva così tanto a riempirsi che egli decise di prendere un campione del disegno visualizzando un pixel ogni otto nella direzione orizzontale e in

quella verticale. In forma compressa, l'immagine risultante non assomigliava all'originale; Braun aveva cambiato la dimensione della griglia, alterando in questo modo la figura d'interferenza.

Otto Smith di Port Townsend, Washington, ha modificato notevolmente la configurazione moiré variando semplicemente lo schema dei colori. Come altri lettori, Smith ha scelto formule diverse da quella di Connett. Certe somme o prodotti di semplici funzioni trigonometriche, per esempio, producono immagini agitate e vorticose che ricordano i disegni a inchiostro colorato usati per abbellire le rilegature interne dei libri pregiati. Smith rileva che figure moiré e altri effetti di interferenza si presentano con regolarità nella grafica al calcolatore. In questo contesto sono noti con il nome di *aliasing*: la tendenza alla comparsa di immagini non volute quando viene digitalizzata una raffigurazione contenente variazioni regolari molto fini.

c

Astrazione maya dal programma per tappezzeria di Smith

Caricature al calcolatore e strani viaggi nello spazio dei volti

di A. K. Dewdney

Le Scienze, dicembre 1986

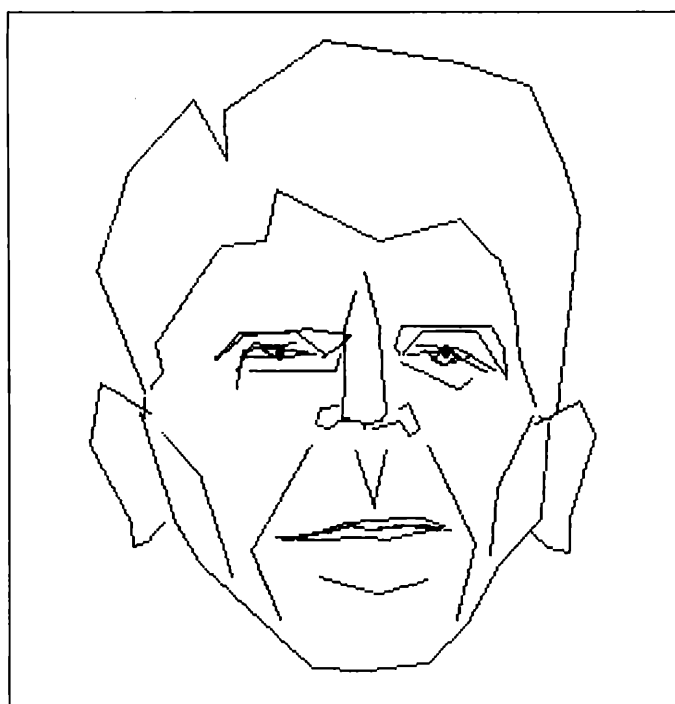
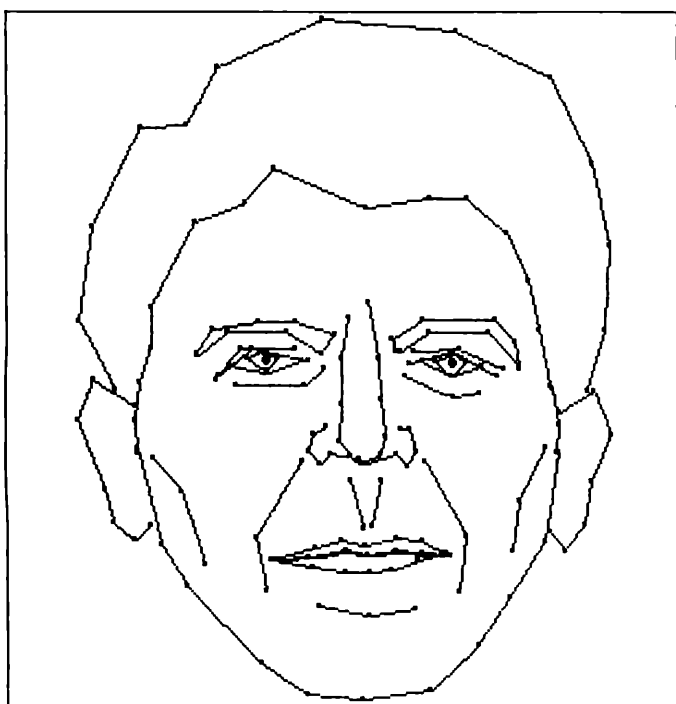
Il volto è inconfondibile: orecchie basse e flosce, zigomi prominenti, ciuffo alto. La faccia di Ronald Reagan è conosciuta in tutto il mondo, ma in un certo senso è ancora più facile riconoscerne le fattezze in una caricatura che in una fotografia. Sicuramente, l'arte della caricatura richiede la capacità di penetrare in profondità nella natura umana. Se così è, il calcolo c'entra ben poco e il calcolatore può essere solo un banale assistente - poco più di un blocco per gli schizzi - che si limita a immagazzinare in forma visiva le sottili interpretazioni del caricaturista.

Oppure no? Le caricature di queste due pagine e delle successive sono state tutte generate da un programma ideato da Susan E. Brennan, una scienziata che lavora nei laboratori della Hewlett-Packard di Palo Alto, in California. Per utilizzare più comodamente il programma potrebbero essere utili un «mouse», una penna luminosa o qualche altro analogo di una matita, ma questi strumenti non sono essenziali. I risultati non dipendono quasi per niente da una mano ferma o da un occhio esercitato. Una volta introdotta nel calcolatore una riproduzione fotografica del volto, il programma entra in azione e disegna la caricatura. Come fa? Detto in breve, la risposta è apparentemente semplice: il programma confronta la fotografia del volto in esame con un volto «medio» che è conservato nella memoria del calcolatore. Le caratteristiche che si differenziano maggiormente dal volto medio vengono ingrandite.

Questo programma ha avuto una naturale origine dalla notevole abilità di Susan Brennan come caricaturista e dal suo interesse per i processi cognitivi che sottendono il riconoscimento di un volto. Questi processi hanno sconcertato a lungo gli psicologi e gli studiosi dei meccanismi cognitivi, e le caricature sembrano svolgere un ruolo particolare nel processo, perché il loro riconoscimento, quando avviene, è quasi istantaneo. Può essere che, invece di ricordare il volto di un amico, ne ricordiamo una caricatura? Per affrontare questi problemi, la Brennan ha inventato la sua semplice tecnica per generare caricature e l'ha descritta nella sua tesi di dottorato al Massachusetts Institute of Technology. Nel tempo libero continua a coltivare questo interesse, mentre nelle ore lavorative si occupa ora della sperimentazione di nuove forme di comunicazione tra uomo e calcolatore che si basano in parte sulla comprensione del linguaggio naturale.

Da un punto di vista concettuale, la tecnica di Susan Brennan presenta strette connessioni con un trucco che gli esperti in animazione al calcolatore chiamano *in-betweening* («intercalazione»).

Immaginate due disegni di oggetti familiari, per esempio una mela e una banana, entrambi ottenuti collegando dei punti con linee (*si veda l'illustrazione di pagina 126*). Ogni punto della mela viene poi messo in corrispondenza con un punto della banana. Se si tracciano linee di collegamento fra le coppie di punti corrispondenti, i punti medi delle linee disegnano una nuova varietà di frutto che è un compromesso tra la mela e la banana - una bananela, ovviamente.

Le stesse linee che collegano la mela e la banana danno anche origine a una forma estrema della banana - dal punto di vista, per così dire, della mela. Si prolunghi ogni linea, oltre la banana, per metà della sua lunghezza originale e si segnino dei punti all'estremità delle linee. Collegando i punti, emerge una caricatura della banana. Analogamente, proiettando le linee di collegamento oltre la mela, si ottiene la caricatura di una mela - dal punto di vista della banana.

Dal realismo al «non volto», passando per due caricature di Ronald Reagan,

Per i volti, si può seguire in buona parte lo stesso procedimento. Ogni coppia di volti definisce due mutue caricature. Le caricature migliori, però, escono dal confronto con un volto medio, appartenente alla norma.

Le norme, nel programma di Susan Brennan, sono ricavate da insiemi di decine di volti reali appartenenti a una base di dati costituita da parecchie centinaia di volti. Vengono scelti dei punti che delineino le caratteristiche di ogni volto e questi punti vengono contrassegnati rispetto a un sistema di coordinate basato su una matrice. L'origine è in alto a sinistra nel piano dell'immagine e i valori delle coordinate crescono procedendo verso destra e verso il basso. La scala è tale per cui la pupilla sinistra si trova nel punto (135,145) e quella destra è nel punto (190,145). Le coordinate di ciascun punto nella norma si ottengono facendo la media dei punti corrispondenti su ciascun volto della base di dati. Per esempio, le coordinate combinate dell'angolo esterno del sopracciglio sinistro danno la coordinata media per l'angolo esterno del sopracciglio sinistro del volto medio. In questo modo si costruiscono tre norme: un volto medio maschile, un volto medio femminile e un anonimo volto medio dall'aspetto un po' androgino, che rappresenta la norma per la maggior parte delle caricature.

Per disegnare una caricatura basata sulla norma, bisogna fornire al programma una versione digitalizzata di un volto reale. In pratica si parte da una fotografia e il programma chiede all'utente di definire, in successione, i 186 punti chiave sulla fotografia. Per esempio, quando il programma richiede i sei punti che costituiscono il sopracciglio sinistro, l'utente può rispondere puntando con un «mouse» a punti successivi sul sopracciglio sinistro dell'immagine fotografica sullo schermo.

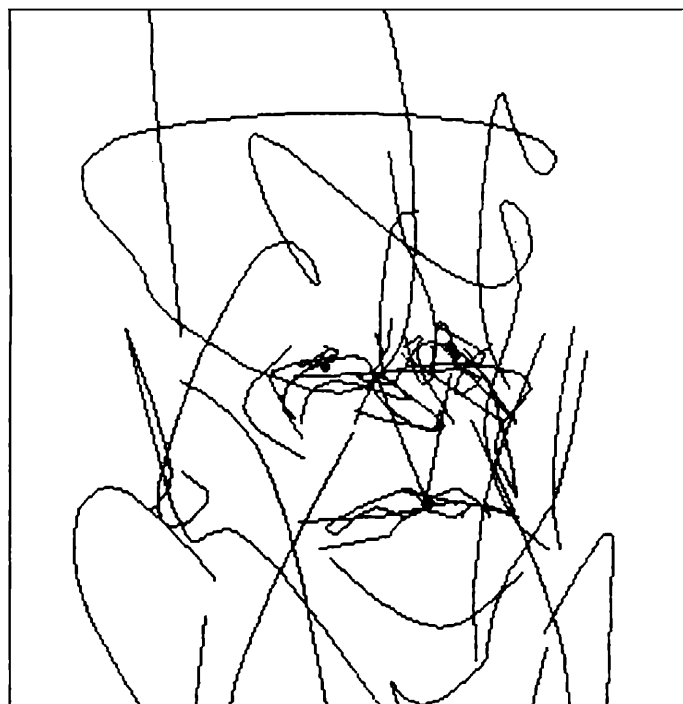
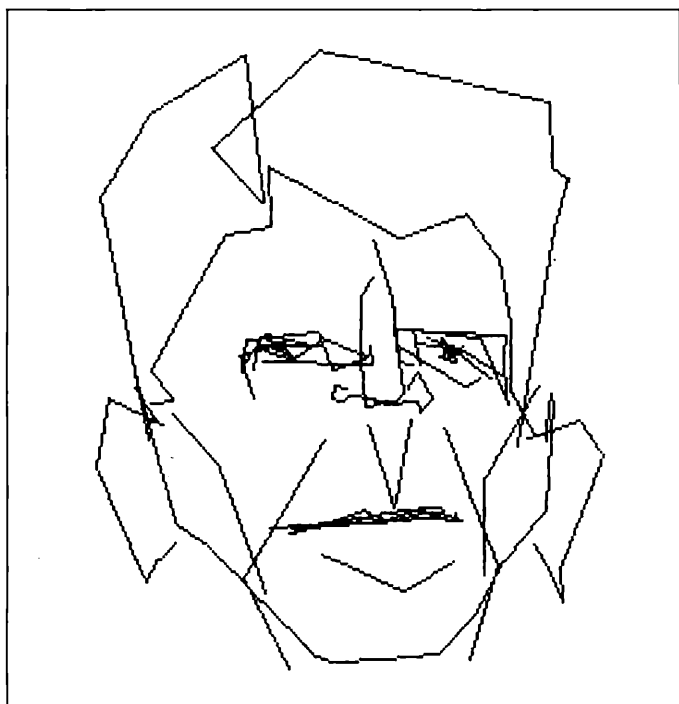
Conviene pensare a questo programma come a una rapida navetta per esplorare quello che la Brennan chiama lo spazio dei volti. Le coordinate immesse per i punti che definiscono una fotografia si possono concatenare in un ordine predeterminato. Ne risulta una lista di numeri che possono essere considerati le coordinate di un singolo punto appartenente a uno spazio a molte dimensioni. Per esempio, sia i volti medi, sia quello fotografico sono rappresentati da 186 punti, ciascuno dei quali ha due coordinate. L'elenco di 372 numeri che ne risulta rappresenta un punto in uno spazio a 372 dimensioni. In linea di principio, a ogni volto si può far corrispondere un punto nello spazio dei volti e ogni coppia di volti nello spazio dei volti può essere congiunta da una linea retta.

Non bisogna, però, lasciarsi ingannare dal concetto di spazio di dimensioni superiori: lo spazio dei volti è semplicemente una comoda astrazione per descrivere differenze e somiglianze tra i volti. I concetti familiari di linea retta e di distanza tra due punti hanno semplici corrispondenti in uno spazio di dimensioni superiori. Tutti i punti disposti in linea retta nello spazio dei volti rappresentano variazioni proporzionali di ciascuna coordinata. La distanza tra due punti nello spazio dei volti misura la loro somiglianza: volti simili sono molto vicini, mentre volti dissimili sono lontanissimi nello spazio dei volti.

Nello spazio dei volti si può immaginare che la norma si trovi vicino al centro di un nugolo di punti che rappresentano immagini realistiche di volti reali. Una linea congiunge ciascun volto reale alla norma. I punti lungo la linea corrispondono a una successione di volti intermedi che assomigliano sempre di più al volto reale. Oltre il volto reale si trovano le caricature, ma esiste un limite naturale per un'accentuazione caricaturale riconoscibile: le caricature alla fine perdono le loro proprietà umane e degenerano in uno stato caotico che Susan Brennan chiama «non volto».

L'idea che ogni volto è un punto in uno spazio dei volti suggerisce un'altra affascinante trasformazione. Dato che, nello spazio dei volti, ogni coppia di volti può essere congiunta da una linea retta, si può chiedere al programma di generare una successione di passaggio da un volto all'altro. Susan Brennan trova particolarmente interessanti queste successioni, quando i due volti terminali sono uno maschile e l'altro femminile: il programma trasforma senza sforzo Elizabeth Taylor, per esempio, in John F. Kennedy.

Il lettore può riprodurre alcuni capolavori della caricatura di Susan Brennan scrivendo una versione ridotta del suo programma, che io chiamo FACEBENDER. L'utente deve inserire almeno due volti: un volto norma e il volto obiettivo, di cui si vuole la caricatura. Ho già parlato in precedenza della norma, le cui coordinate sono state generosamente fornite dalla Brennan (*si veda l'illustrazione in alto a pagina 127*). L'utente deve allora trasformare la faccia obiettivo nella stessa forma. Se non possiede raffinate apparecchiature di digitalizzazione, il lettore può ugualmente trasformare la fotografia di un suo caro (magari se stesso) in una lista di coordinate. La Brennan raccoman-

in FACEBENDER, un programma basato sul lavoro di Susan E. Brennan

da, tuttavia, che il volto della fotografia abbia un'espressione neutra: anche un debole sorriso si trasformerà in una smorfia mostruosa. Il volto, inoltre, deve essere completamente frontale; se la testa è voltata, FACEBENDER ne provocherà una torsione molto più accentuata.

Per determinare la scala degli assi, presupponete che le coordinate delle pupille destra e sinistra siano le stesse di quelle della norma: la sinistra dovrebbe essere in (135,145) e la destra in (190,145). (Ricordate che le coordinate orizzontali aumentano da sinistra a destra, mentre quelle verticali dall'alto al basso.) Una volta stabilita la scala di distanza, l'utente deve compiere un'accurata misurazione al fine di trovare le altre coordinate. Nello schema di digitalizzazione di Susan Brennan i punti sul volto sono organizzati in 39 caratteristiche facciali; ciascuna è una successione di punti connessi. L'ordine dei punti dipende dall'orientazione: per lineamenti che si sviluppano principalmente in orizzontale i punti sono elencati da sinistra a destra, per quelli che invece si sviluppano principalmente in verticale i punti sono elencati dall'alto in basso.

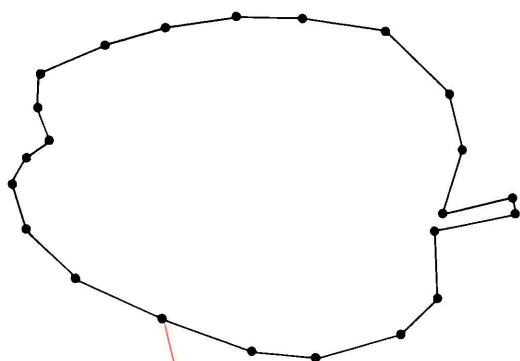
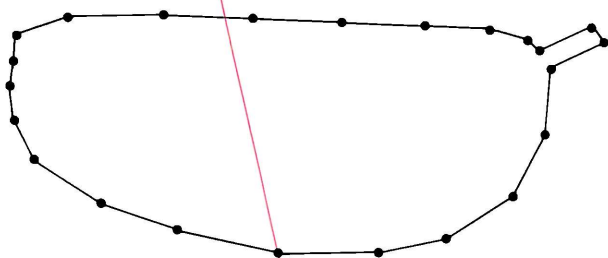
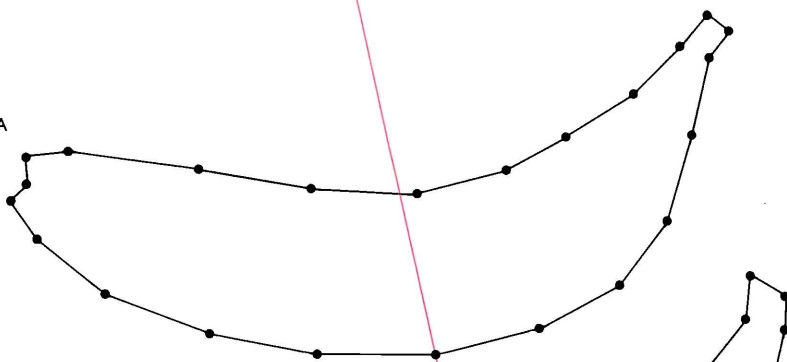
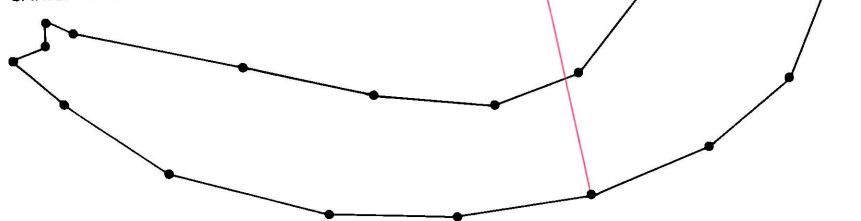
Come trasformare una mela in una banana, e oltre

La Brennan riconosce che l'identificazione dei punti chiave di un volto procede principalmente per tentativi ed errori; si può, tuttavia, usare come guida il volto del presidente Reagan. Per questa ragione è importante che sia la stessa persona a eseguire la trasformazione dalla fotografia all'elenco dei numeri per ciascun volto immesso in una base di dati.

FACEBENDER immagazzina i due volti digitalizzati in matrici chiamate *volto* e *norma*. Per creare una visualizzazione è necessaria una terza matrice chiamata *vis*. Tutte e tre le matrici hanno 186 righe e due colonne: un punto del volto per riga e una coordinata per colonna. I punti sono sistemati nell'ordine seriale dato nella lista per *norma*. Il vantaggio di questo ordinamento sta nel fatto che tutte le linee della rappresentazione finale si possono in tal modo tracciare fra punti successivi della matrice; ovviamente questo non avviene quando una caratteristica è completa e se ne deve iniziare un'altra.

Il primo lineamento che il programma disegna è la pupilla dell'occhio sinistro; il secondo è la pupilla dell'occhio destro. Ciascuna pupilla si può rendere sia con un punto, sia con un piccolo cerchio; i cerchietti hanno un aspetto più amichevole. Per le caratteristiche rimanenti si tracciano linee per congiungere punti consecutivi della matrice. Si rende necessaria una matrice speciale chiamata *caratteristiche* per omettere la linea tra l'ultimo punto di una caratteristica facciale e il primo punto della caratteristica successiva. La matrice dà il numero di punti per ciascuna caratteristica e un doppio ciclo sovrintende alle omissioni (*si veda l'illustrazione in basso nella pagina a fronte*).

Poiché sono state disegnate le prime due caratteristiche, il sottoprogramma di visualizzazione comincia con la terza caratteristica e precisamente l'iride sinistra. Il primo punto nell'iride sinistra è il terzo punto della matrice *vis*, che è indicizzata dalla variabile *i*, il cui valore iniziale è, quindi, posto uguale a 3. La matrice *caratteristiche* è indicizzata da un'altra variabile, *j*, che varia da 1 a 37, perché ci sono ancora 37 caratteristiche da disegnare. All'interno del ciclo di *j* un'altra variabile chiamata *contatore* tiene il conto del numero di linee tracciate per ciascuna caratteristica; essa aumenta di 1 a ogni passaggio per il ciclo controllato da *j*. Anche l'indice *i* aumenta a ogni passaggio per il ciclo; esso identifica il punto della matrice *vis* che in quel momento è impegnato nel frenetico esercizio di congiungere i punti.

All'interno del ciclo controllato da *j* c'è un secondo ciclo, chiamato «ciclo *while*», che confronta il numero di punti congiunti fino a quel momento nella caratteristica *j* con il numero totale di punti di quella caratteristica. Il programma abbandona il ciclo *while* quando i due numeri sono uguali; la caratteristica è completa. Se vi sono ancora punti da congiungere nella caratteristica, il programma disegna una linea dal

punto *i* al punto *i* + 1 della matrice *vis*. La mia notazione è solo una forma stenografica: un vero e proprio comando di visualizzazione richiederebbe una linea a partire dal punto di coordinate *vis*(*i*,1) e *vis*(*i*,2) fino al punto di coordinate *vis*(*i* + 1,1) e *vis*(*i* + 1,2).

Il cuore di FACEBENDER è il suo sottoprogramma di accentuazione, la cui struttura è ancora più semplice di quella del sottoprogramma di visualizzazione che ho appena descritto (*si veda l'illustrazione in basso*). Per ciascuno dei 186 punti facciali delle matrici *volto* e *norma*, il ciclo calcola una nuova matrice detta *distorsione*, che codifica la caricatura da ottenere. Ogni coordinata della matrice *distorsione* viene calcolata aggiungendo la coordinata corrispondente della matrice *volto* a una quantità che accentua fortemente le differenze tra *norma* e *volto*. Il fattore di accentuazione *f* viene inserito dall'utente; poi vengono moltiplicate per *f* la differenza tra le coordinate orizzontali di *volto* e *norma* e anche la differenza tra le coordinate verticali.

Resta soltanto da organizzare il programma e, volendo, perfezionare il sottoprogramma di disegno. Una semplice tecnica non procedurale è quella di sistemare sia il sottoprogramma di visualizzazione sia quello di accentuazione all'interno di un ciclo interattivo che domanda all'utente: «Vuoi provare ancora?». Il programma deve anche sollecitare all'utente il fattore di accentuazione. Disponete l'invito del sistema in modo che si possano sperimentare parecchi fattori di accentuazione diversi senza dover immettere nuovamente tutta la matrice *volto*; in tal modo è facile confrontare il loro effetto sulla caricatura.

Si possono migliorare un po' i disegni se si uniscono i punti con linee curve chiamate, con termine inglese, *spline*, invece che con rette. Le curve evitano gli zig zag e congiungono con continuità i punti. Il programma della Brennan generalmente disegna spline per formare i contorni regolari dei lineamenti facciali. Mi sono reso conto subito, però, che le spline sarebbero risultate difficili da spiegare in una rubrica dedicata soprattutto a programmi facili e allora ho chiesto a Susan Brennan se esistesse un metodo alternativo. Si potevano usare anche le linee rette? Con sorpresa sua e mia, le caricature disegnate con linee rette erano valide quasi quanto quelle disegnate con le curve. In effetti, tutte le immagini che compaiono in questo articolo sono state disegnate usando linee rette. Con una piccola perdita estetica il programmatore può evitare l'impiego di una tecnica più problematica. Si può subito cominciare con il sottoporre alla digitalizzazione la propria fotografia preferita.

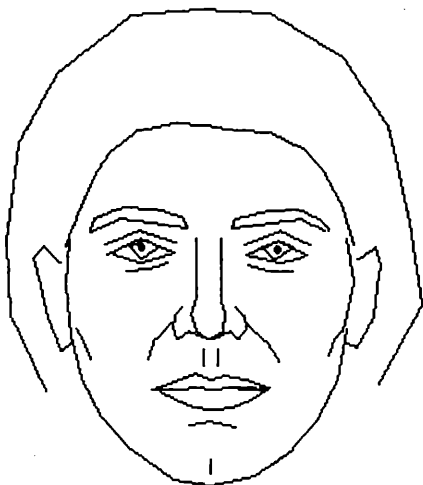
Il generatore di caricature di Susan Brennan è stato applicato in numerosi studi sul riconoscimento dei volti. Volti prodotti dal suo programma sono stati trasmessi via cavo telefonico come parte di un esperimento di teleconferenza organizzato dal Media Laboratory del Massachusetts Institute of Technology. L'anno

PUPILLA SINISTRA	1 PUNTO	(135,145)
PUPILLA DESTRA	1 PUNTO	(190,145)
IRIDE SINISTRA	5 PUNTI	(134,141) (128,144) (133,149) (140,144) (135,141)
IRIDE DESTRA	5 PUNTI	(190,141) (184,144) (189,149) (196,144) (190,141)
PALPEBRA SINISTRA IN BASSO	3 PUNTI	(119,147) (133,140) (147,146)
PALPEBRA DESTRA IN BASSO	3 PUNTI	(177,147) (190,141) (203,147)
OCCHIO SINISTRO IN BASSO	3 PUNTI	(121,147) (133,150) (147,146)
OCCHIO DESTRO IN BASSO	3 PUNTI	(177,147) (191,150) (201,148)
OCCHIO SINISTRO IN ALTO	3 PUNTI	(118,143) (132,137) (148,142)
OCCHIO DESTRO IN ALTO	3 PUNTI	(176,143) (191,137) (204,143)
LINEA DELL'OCCHIO SINISTRO	3 PUNTI	(127,154) (135,153) (144,150)
LINEA DELL'OCCHIO DESTRO	3 PUNTI	(178,151) (187,154) (196,154)
LATO SINISTRO DEL NASO	6 PUNTI	(156,140) (156,153) (156,165) (154,172) (156,179) (161,182)
LATO DESTRO DEL NASO	6 PUNTI	(166,140) (166,153) (166,166) (168,172) (167,179) (161,182)
NARICE SINISTRA	6 PUNTI	(150,169) (147,173) (146,178) (148,182) (153,179) (161,182)
NARICE DESTRA	6 PUNTI	(173,169) (176,172) (177,178) (174,182) (170,179) (163,182)
SOPRACCIGLIO SINISTRO IN ALTO	6 PUNTI	(112,137) (113,132) (125,127) (139,128) (150,131) (152,136)
SOPRACCIGLIO DESTRO IN ALTO	6 PUNTI	(171,136) (173,132) (186,129) (199,128) (208,132) (211,137)
SOPRACCIGLIO SINISTRO IN BASSO	4 PUNTI	(112,138) (124,132) (138,134) (152,136)
SOPRACCIGLIO DESTRO IN BASSO	4 PUNTI	(171,136) (187,134) (200,132) (210,137)
LABBRO SUPERIORE IN ALTO	7 PUNTI	(137,203) (149,199) (156,196) (162,199) (168,197) (177,199) (187,202)
LABBRO SUPERIORE IN BASSO	7 PUNTI	(138,203) (148,203) (156,202) (163,203) (170,202) (178,203) (186,202)
LABBRO INFERIORE IN ALTO	7 PUNTI	(138,203) (149,203) (156,202) (163,203) (170,202) (177,202) (186,203)
LABBRO INFERIORE IN BASSO	7 PUNTI	(141,204) (148,207) (155,210) (163,211) (171,210) (179,207) (185,203)
LATO SINISTRO DEL VOLTO	3 PUNTI	(103,141) (101,160) (104,181)
LATO DESTRO DEL VOLTO	3 PUNTI	(219,140) (222,159) (218,179)
ORECCHIO SINISTRO	7 PUNTI	(99,150) (92,144) (88,149) (90,160) (94,174) (99,187) (104,184)
ORECCHIO DESTRO	7 PUNTI	(224,149) (231,144) (234,151) (232,160) (230,173) (224,185) (219,184)
MASCELLA	11 PUNTI	(104,181) (108,199) (115,214) (129,228) (147,240) (162,243) (180,239) (196,228) (207,215) (215,199) (219,178)
ATTACCATURA DEI CAPELLI	13 PUNTI	(101,144) (107,129) (114,114) (120,104) (131,95) (146,92) (160,93) (174,95) (188,96) (201,103) (210,114) (217,126) (222,143)
PARTE SUPERIORE DELLA TESTA	13 PUNTI	(93,204) (78,173) (76,142) (82,101) (99,70) (129,46) (158,44) (188,45) (217,64) (236,94) (245,134) (250,168) (233,200)
LINEA DELLA GUANCIA SINISTRA	3 PUNTI	(145,175) (139,182) (135,190)
LINEA DELLA GUANCIA DESTRA	3 PUNTI	(178,176) (185,183) (190,191)
ZIGOMO SINISTRO	3 PUNTI	(105,178) (109,184) (112,190)
ZIGOMO DESTRO	3 PUNTI	(218,178) (214,183) (211,189)
LINEA SINISTRA DEL LABBRO SUPERIORE	2 PUNTI	(159,186) (159,193)
LINEA DESTRA DEL LABBRO SUPERIORE	2 PUNTI	(165,186) (165,193)
FESSURA DEL MENTO	2 PUNTI	(162,232) (162,238)
LINEA DEL MENTO	3 PUNTI	(153,218) (162,216) (173,219)

Le coordinate per i punti di un volto medio

```
SOTTOPROGRAMMA DI VISUALIZZAZIONE
i ←2
for j = 1 to 37
    i ← i + 1
    contatore ← 1
    while contatore < caratteristiche (j)
        disegna linea da vis(i) a vis (i + 1)
        contatore ← contatore + 1
        i ← i + 1

SOTTOPROGRAMMA DI ACCENTUAZIONE
for i = 1 to 186
    distorsione (i,1) ←volto (i,1) + f x[volto (i,1) −norma (i,1)]
    distorsione (i,2) ← volto (i,2) + f x [volto (i,2) − norma (i,2)]
```

***Il cuore di* FACEBENDER**

Il volto medio androgino

molto accentuate venivano riconosciute, il riconoscimento avveniva in modo significativamente *più veloce* - circa due volte più in fretta, in effetti, dei disegni realistici delle stesse persone.»

Susan Brennan propone un gran numero di altri esperimenti con il generatore di caricature. Sarebbe affascinante, per esempio, individuare la «norma» assunta dai caricaturisti. Ricevuta la caricatura di un dato soggetto da un certo artista, la ricercatrice cercherebbe di ripercorrere all'indietro l'accentuazione fino a determinare il volto normale, presumibilmente riposto da qualche parte nell'inconscio dell'artista, da cui è derivata l'accentuazione. A questo punto è possibile porsi una serie molto interessante di quesiti. La norma ricostruita sarebbe quasi uguale passando da un soggetto a quello successivo? Artisti diversi assumono norme diverse?

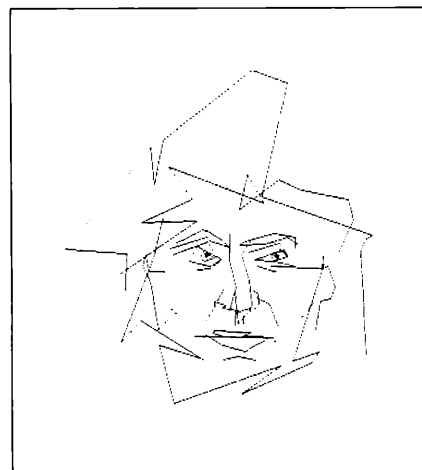
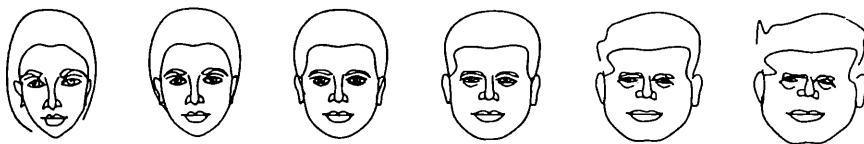
Autocaricatura del volto medio

scorso Susan Brennan ha effettuato un esperimento in collaborazione con Gillian Rhodes, dell'Università di Otago in Nuova Zelanda, allora laureanda con Roger N. Shepard della Stanford University. In primo luogo sono state generate caricature di membri della facoltà e di studenti di psicologia di Stanford; queste caricature sono state poi contrapposte a normali disegni in un test di riconoscibilità.

Ecco come Susan Brennan ha riassunto i risultati: «Il generatore di caricature era particolarmente utile per questo studio perché ci metteva in grado di generare stimoli che variano in modo continuo e controllato; i precedenti studi sulla percezione dovevano confrontare caricature con fotografie o altri tipi di immagini non così simili, risultando quindi non liberi da effetti rappresentativi. Si è trovato che le caricature non risultano particolarmente *migliori* come rappresentazioni riconoscibili (le rappresentazioni «migliori» erano solo leggermente accentuate), ma quando le caricature

Soluzioni proposte

Nell'articolo del dicembre scorso ho descritto un programma, chiamato FACEBENDER, ispirato al lavoro di Susan E. brennan dei laboratori della Hewlett-Packard di Palo Alto, California. In ingresso il programma accetta la versione digitale di una faccia, di cui si vuol disegnare la caricatura; faccia che poi confronta con un volto medio di riferimento, analogamente digitalizzato, che ha in memoria. Il programma distorce poi ciascuna caratteristica del volto in ingresso in misura proporzionale alla distanza dalla corrispondente caratteristica del volto di riferimento; un orecchio moderatamente grande rispetto a quello di riferimento verrà ulteriormente allargato moltiplicando tutte le differenze per un fattore di accentuazione f.

I lettori che vogliano realizzare il programma FACEBENDER possono forse rimanere spaventati dalla prospettiva di digitalizzare il proprio volto a partire da una fotografia. Pat Macaluso di White Plains, New York, usa il volto di riferimento come base per la propria caricatura. «La chiave - dice Macaluso - sta nel proporzionare l'entità della variazione alla dimensione di ciascuna caratteristica. Così un orecchio riceve una variazione assoluta maggiore di quella della fossetta nel mento. Basta calcolare il "riquadro" in grado di contenere ciascuna caratteristica calcolando il massimo e il minimo delle coordinate x e y per ciascuna caratteristica.» All'interno di questa cornice la quantità di distorsione è governata da numeri casuali scelti dal programma. In questo modo si può produrre una varietà infinita di volti con la versione ad autoriferimento di FACEBENDER progettata da Macaluso. Una delle caricature prodotte con questa tecnica assomiglia a Leonardo da Vinci. La si può vedere qui sopra.

Un lettore noto soltanto come DMI di Pasadena, California, ha avanzato una proposta per evitare «l'assenza di volto», il temuto stato che si verifica quando il fattore di accentuazione è troppo grande; tutte le caratteristiche degenerano in un miscuglio di poligoni disordinato e irriconoscibile. Immaginiamo che il volto di cui si deve fare la caricatura sia sovrapposto al volto di riferimento e che i punti di riferimento siano collegati da molle. Il processo di distorsione cerca di spostare i punti del volto in ingresso, ma così facendo incontra la resistenza delle molle. Difficilmente si incide quindi sulle piccole distorsioni, ma si evita che quelle grandi portino allo stato senza volto.

Elizabeth Taylor (nella parte di Cleopatra) si incontra con l'ex presidente Kennedy nello spazio dei volti

IV. Giocare con il linguaggio

Il linguaggio è il più raffinato fra gli strumenti di comunicazione di cui l'evoluzione ci abbia dotati: con le parole possiamo dare notizie, esporre fatti, impartire ordini, chiedere informazioni, esprimere desideri, commuovere, insultare, lusingare, condannare...; nessuno degli altri modi di comunicazione della specie umana ha altrettanta duttilità e altrettanto potere espressivo. A dispetto della molteplicità delle lingue, ogni bambino (salvo poche eccezioni patologiche) impara rapidamente a usare questo strumento, e c'è chi riesce a padroneggiarlo tanto bene da fare del suo uso, parlato o scritto, l'occupazione della propria vita (o quantomeno un eccellente mezzo di sostentamento). Eppure, nonostante questo e nonostante che il linguaggio sia parte della nostra dotazione da migliaia, se non milioni di anni, accade per il linguaggio più o meno quel che accade per la visione e il riconoscimento delle forme: sappiamo, del funzionamento sottostante a questi processi, molto poco. Non sempre ne siamo consapevoli, ma la cosa appare in bella evidenza proprio quando ci si pone il problema di «insegnare» questi processi (linguistici o percettivi che siano) a una macchina.

I «linguaggi» con cui per ora parliamo a un calcolatore sono, in effetti, una ben misera imitazione del linguaggio naturale: rigidi, poco espressivi, noiosi. E se si pensa, per esempio, alla rapidità con cui si è evoluta, nel giro di pochi decenni, la tecnologia dei mezzi di trasporto, dall'invenzione dell'automobile al Concorde, non può non far riflettere la lentezza con cui, invece, si sta evolvendo la tecnologia delle interfacce tra calcolatori e utenti. Non è solo un problema di scarsa attenzione dei «tecnici» per questi argomenti (anche questo, certo, può aver avuto qualche influenza): si direbbe che qui siano in gioco questioni davvero molto grosse, forse anche aspetti che ci sfuggono ancora in gran parte.

I tre articoli riportati in questa ultima parte del volume scalfiscono soltanto in qualche punto il blocco dei problemi che il linguaggio ancora ci riserva: vista la scarsità di conoscenze, però, il gioco qui, più ancora che in altre direzioni, può rivelarsi un'eccellente forma di esplorazione. La posta non è piccola: i calcolatori hanno ormai posti di rilievo in molte attività cruciali e la capacità di interagire verbalmente o per iscritto in linguaggio naturale, quindi con immediatezza espressiva e prontezza di riflessi, in situazioni normali può fare la differenza fra un lavoro di buona o di cattiva qualità, ma in situazioni estreme può fare la differenza fra il successo e l'insuccesso.

Qui, in effetti, l'attenzione si concentra solo sul linguaggio scritto: il parlato coinvolge problemi ulteriori, parte dei quali riguardano il campo dell'acustica. Ma si possono vedere lo stesso alcuni punti cruciali, ed emerge chiaramente come la produzione e la comprensione del linguaggio chiamino in causa direttamente la conoscenza e la sua rappresentazione. L'ultimo articolo sulla pazzia artificiale, poi, può essere visto in quella tradizione di ricerca scientifica (che non poco ha dato alla biologia) che considera lo studio del «patologico» come una via privilegiata per la comprensione della «normalità».

Un rapporto di ricerca sulla sottile arte del trasformare letteratura in non senso

di Brian Hayes

Le Scienze, gennaio 1984

Quasi tutti i programmi per calcolatore possono portare a risultati senza senso se le informazioni su cui debbono lavorare sono sufficientemente confuse. Il principio è ormai talmente assodato da rendere superflua un'altra dimostrazione: è questo il senso dell'espressione «*garbage in, garbage out*» (come dire «più spazzatura entra, più spazzatura esce»). È comunque possibile, con un po' di attenta applicazione, creare un programma che accetti in ingresso grandi capolavori della letteratura e arrivi a produrre in uscita delle totali assurdità. Si immette l'ultimo atto del *Macbeth* ed esce una storia raccontata da un idiota, piena di suoni deliranti, priva di significato. *Questa* è oggi l'elaborazione dei dati. (La trasformazione inversa, ahimé, si direbbe molto più difficile.)

Il passaggio da letteratura a discorso farfugliato avviene in due stadi. Dapprima un testo viene «letto» dal programma e ne vengono estratte e registrate determinate proprietà statistiche. Tali proprietà definiscono la probabilità che una certa lettera segua un'altra lettera, o un'altra sequenza di lettere, nel testo di partenza. Nel secondo stadio, si genera un nuovo testo scegliendo le lettere a caso secondo le probabilità registrate. Il risultato è un flusso di caratteri che riproduce le proprietà statistiche del testo originale ma il cui eventuale significato, ammesso che ne abbia uno, è del tutto accidentale.

Non riesco a immaginare un metodo di imitazione più rozzo. Nel programma non esiste la benché minima rappresentazione del concetto di una parola, men che meno un'indicazione di ciò che le parole potrebbero significare. Non c'è rappresentazione di una struttura linguistica più elaborata di una sequenza di lettere. Il testo risultante è il più sgraziato dei *pastiche,* in cui vengono conservate solo le caratteristiche più superficiali del testo originale. Ciò che è notevole è che questo semplice esercizio produce a volte qualcosa con un aspetto sorprendentemente familiare. È un *nonsense,* ma non indifferenziato; lo si direbbe piuttosto un nonsense chauceriano o shakespeariano o jamesiano. In effetti, una volta eliminato del tutto il contenuto semantico, ciò che risulta più evidente è la modalità stilistica. Viene da chiedersi: quanto sono vicine alla superficie le caratteristiche che definiscono lo stile di un autore?

Il processo per generare prosa casuale è stato dettagliatamente studiato da William Ralph Bennett, Jr., della Yale University. Le proprietà statistiche del linguaggio hanno occupato un posto di rilievo in un suo corso sulle applicazioni del calcolatore e l'argomento ha anche notevole spazio nel suo manuale di introduzione alla programmazione, *Scientific and Engineering Problem-solving with the Computer.* (Il libro è molto più vivace di quanto il titolo potrebbe far pensare. Tra i problemi presi in considerazione vi sono la partita di football Princeton-Dartmouth giocata nel 1950 nel pieno di un uragano, la diffusione della sifilide in una popolazione di marinai e prostitute e un'analisi spettrale del suono del cromorno, dell'oboe e della «canna per innaffiare a modo bloccato».)

Bennett rileva che i primi riferimenti noti alla generazione casuale del linguaggio si trovano in *Maxims and Discours* di John Tillotson, arcivescovo di Canterbury verso la fine del XVII secolo. Difendendo la creazione divina Tillotson scriveva: «Quante volte un uomo dovrebbe lanciare per terra delle lettere, che tiene mischiate in una borsa, prima che esse si dispongano a formare una poesia, un'affermazione o un buon discorso in prosa? E un piccolo libretto non può essere fatto dal Caso con altrettanta facilità di questo grande Volume del Mondo?»

Per più moderne considerazioni sul linguaggio casuale, il punto di partenza è la seguente affermazione espressa da Sir Arthur Eddington nel 1927: «Se un esercito di scimmie pestasse su delle macchine da scrivere, *potrebbe* scrivere tutti i libri del British Museum.» Anche Eddington voleva evidenziare l'improbabilità di un simile esito; lo citava come esempio di evento che potrebbe accadere in linea di principio ma che in pratica non accade mai. Malgrado tutto, dai tempi di Eddington la possibilità di trovare dei veri geni in questa scimmiesca produzione casuale ha assunto una sua propria vita letteraria. Bennett cita i lavori di Russell Maloney e Kurt Vonnegut, Jr., e un lavoro da cabaret di Bob Newhart.

Il processo immaginato da Eddington può essere simulato da un programma che chiamerò generatore di testo d'ordine zero. Prima di tutto si decide un alfabeto, ovvero un insieme di caratteri, per stabilire quali tasti mettere sulle macchine per scrivere delle scimmie. In alcune simulazioni d'ordine superiore diviene importante ridurre al minimo il numero di simboli e per coerenza sembra meglio adottare lo stesso insieme di caratteri nell'ordine zero. Ho quindi seguito la raccomandazione di Bennett di scegliere un insieme di 28 simboli: le 26 lettere maiuscole, lo spazio tra parole (che il calcolatore considera un carattere come gli altri) e l'apostrofo (che nell'inglese scritto è più comune delle tre o quattro lettere meno comuni).

La scimmia ideale, priva di indicazioni, avrebbe in ogni momento la stessa probabilità di battere un qualsiasi tasto. Questo comportamento può essere simulato da una semplice strategia. A ogni carattere dell'insieme prescelto si assegna un numero da zero a 27. Per ogni carattere da generare si sceglie a caso un intero appartenente allo stesso dominio e si stampa il carattere corrispondente. Nella figura di questa pagina si può vedere un piccolo esempio di testo creato con questa procedura: non presenta alcuna somiglianza né con l'inglese scritto né con qualsiasi altra lingua. Le «parole» tendono a essere straordinariamente lunghe (in media sono di 27 lettere) e dense di consonanti. La ragione, naturalmente, è che le frequenze delle varie lettere nel vero testo inglese sono lungi dall'essere uniformi. Il solo spazio tra parole di solito rappresenta circa un quinto dei caratteri, mentre J, Q, X e Z insieme non toccano l'1 per cento. In una simulazione d'ordine zero tutti i caratteri hanno invece la stessa frequenza di 1/28.

Il lavoro comico di Bob Newhart è giocato sugli ispettori che hanno l'incombenza di leggere la produzione delle scimmie. Dopo molte ore di impegno su inintellegibili sequele di lettere, si imbattono nella frase «To be or not to be, that is the gesorenplatz...». In realtà, anche arrivare a questo punto è altamente improbabile; le prime nove parole del monologo di Amleto hanno una probabilità d'uscita pari a una volta ogni 2×10^{46} caratteri. In una distribuzione di 50 000 caratteri mi è

'PWGMMLTHIDVGRHPEDFCXFEKFNOPYPQSXZRUXG'YS'AEEU PEDEGLQYFUWPO'IKI
QTONIXJKZEUKDXWKKJREHYHPKWUJHLEJNBPLQ AIEOQXUBJYYVIFFDPQGIGZNTI
RQXPDJ NQESPQMCRSNGMKQEZICZV'GSWALK ZZEYIBBOTDCRSMK'VI MRCZXUBI
SNEQ'VQQHFQUCBJXZRVVNIBHFJEFTCFJPWFOIYHOMPNFSFWKNCMVLOJJBX
QV KIZTLNRWGGTZFPZPQQCGVJCPAYRDQJRMYSWCGABRXLERCYYRHQCHTOQ'UT
FMRITFTIZUIWTSTXWQGOCAFXJOZYKSTV'BYOBEUFIRQWQ VOUVQJPRKJWBKPLQZCB

Testo casuale d'ordine zero con un alfabeto di 28 simboli

riuscito di trovare una sola occorrenza di TO e un'altra di NOT, a molte righe di distanza una dall'altra. (In realtà non mi sono letto i 50 000 caratteri, ma ho impostato un programma di ricerca.)

Un primo passo per migliorare l'abilità letteraria delle scimmie consiste nell'adeguare la probabilità di scelta di una certa lettera alla sua effettiva frequenza nell'inglese scritto. Il problema consiste nel costruire una macchina per scrivere con, diciamo, 2500 tasti per la spaziatura, 850 tasti E, 700 tasti T e così via. Le frequenze delle lettere potrebbero essere medie calcolate su un ampio campione di prosa inglese, ma è più conveniente e anche più interessante fondarsi su un particolare testo sorgente. Un programma che scelga i caratteri con una tale distribuzione di frequenze è un generatore di testo di primo ordine.

PRIMO ORDINE

HUD T ALONIT NTA SN TVIOET ELERFOAD PE TRLTWTL N CABEG TYLUEMU TIGT
BH OFDRRIC O STU HOOOTO YATNDL UYA HWAE SS NLSDB OTRORT DEERARFT
D LBFF HHARE MW OSPE OFOIT SEOUN GTUMG H N GHKOY T EAOS A SD E TNNE
PEHAGIADIHNATO AATSAGI ED INNE ABRA TAAM GT E TWNO HEWIIGUTNCM GA SFHHY
HREBH RARE OOSY LFE OC EGGTA WIFRTYE EUS DA ETO WF EIT ERNETEBTSTTELO
NTAAN O YEETWNSONRNHN TYHVN NLUESETTHLGEAKPNNMTIA TSM REEANTVONC POE
RUTP EOIT L IEETGTWHSW H KHHER W OLIOEWOEPT D AEYBSTNHGDNPT C TNLINHH
KHHE E RTVIOB EI K EOAFPUTSTTAS NA LAN SRDF D NMTHESKO UGEEDICRAWDT OBD
TUIML WSORGNETE

SECONDO ORDINE

BEGASPOINT IGHIANS JO HYOUD WOUMINN BONUTHENIG SPPRING SBER W IDESE WHE D
OOFOMOUT O CHEDA AFOOIAUDO IS WNY UT DRSASER LD OT POINE ETHAT FOEVEL BE
ORRI IVER BY HE T AS I HET W BE T WAU GIM UTHENTOTETHAVE THIKEWOITOCOUTORE
TATHASTHEE AT D Y WAN TOND SE TEDING US AKIN WING W TE T BO TOTSTHINGATONO
EN T LLY WID OUCOUSIND HEF THIMES AG T BENG LORYE ALLATHOMOFTHER TOUDIMS YS
S ORYRY THERNG S HE M G M ANG S CITOOFO HEN G BEST ONDLOL ANE DO HE
ICISEKERIT ME NKITHADIMUPL WHES HT BATHE T LOR WITULOWAYE WATHEG M
LEROMAUN OUGS POUPO O HASING LIN ON ASHAN AWFAS HET ND MEDE

TERZO ORDINE

MAY THOT TO THER YOURS CHIM JOSE EY EILLY JUSED AND HID YEL THE MARK WASK
TROOFTEN HEREY LING SH THAVERED HER INCED I MEA BUT DAY WOM THE EAKIN WIPS
AS SUGH THE WAY LIARADE TH MY HE ALMASEETIR ANICIOUT JOSIDNTO GRATEVE NO
VER BIGH WER ACCOW WAS I GEORE HENDSO EGGET PUT TO SQUAD TRADE OFF GIN
GO ME HER SPING HE CONE WELL FEWHEY THEYES AND AND QUICE YOULDNT HER
ORL SO MAKING RINGS SOMET DREAVE HISETTO COMAD THAT ME WE MIG TOLD THE
THERFUMBECK OT OFF FEELP HE WAST ITS LETHOTTEN ITHEE ROWN YOURS FEL FOR
SOME IF WIS HE STAKED UPPOIS SHENS NO TILL HIM I WAY SO WHATEALWAS WER TWE
NER DING O THIS IT IN ANIGH ACK REAN THAT DO GETHE BITER

Testo casuale di primo, secondo e terzo ordine, basato sul «Soliloquio di Molly Bloom»

AD CON LUM VIN INUS EDIRA INUNUBICIRCUM OMPRO VERIAE TE IUNTINTEMENEIS
MENSAE ALTORUM PRONS FATQUE ANUM ROPET PARED LA TUSAQUE CEA ERDITEREM IN
GLOCEREC IOVELLUM ET VEC IRA AE DOMNIENTERSUO QUE DA VIT INC PARBEM ETUS
TU MEDE DERIQUORUMIMO PEREPORIDEN HICESSE COSTRATQUIN FATU DORAEQUI POS
PRIENS NOCTA CIENT HUCCEDITAM PET AUDIISEDENDITA QUE GERBILIBATIA VOLAEQUE
ORECURICIT FES ADSUE ARCUMQUE LULIGITO PIMOES PERUM NOSUS HERENS EA
CREPERESEM ETURIBUS AVIS POS AT IS NOMINE FATULCHENTURASPARIS AUDEDET PARES
EXAMENDENT DUM REMPET HA REC ALEVIREM ORBO PIERIS ATAE PARE OCERE RAS

QUALTA 'L VOL POETA FU' OFFERA MAL ME ALE E 'L QUELE ME' E PESTI FOCONT E 'L M'AN
STI LA L' ILI PIOI PAURA MOSE ANGO SPER FINCIO D'EL CHI SE CHE CHE DE' PARDI
MAGION DI QUA SENTA PROMA SAR OMI CHE LORSO FARLARE IO CON DO SE QUALTO
CHE VOL RICH'ER LA LI AURO E BRA RE SI MI PAREMON MORITA TO STOANTRO FERAI TU
GIA FIGNO E FURA PIA BUSCURA QUAND'UN DEL GUARDI MIN SA PAS DELVENSUOLSI PER
MUSCER PIE BRUI TA DORNO TITTRA CHE PO E PER QUE LI RINONNIMPIAL MIN CH'I'
BARVEN TA FUI PEREZZA MOST' IO LA FIGNE LA VOL ME NO L'E CHE 'L VI TESTI CHE
LUNGOMMIR SI CHE FACE LE MARDA PRESAL VOGLJCESA

PONT JOURE DIGNIENC DESTION MIS TROID PUYAIT LAILLE DOUS FEMPRIS ETIN
COMBRUIT MAIT LE SERRES AVAI AULE VOIR ILLA PARD OUR SOUSES LES NIRAPPENT LA
LA S'ATTAIS COMBER DANT IT EXISA VOIR SENT REVAIT AFFRUT RESILLESTRAIS TES FLE
LA FRESSE LES A POURMIT LE ELLES PLOIN DAN TE FOLUS BAIER LA COUSSEMBREVRE
DE FOISSOUR SOUVREPIACCULE LE SACTUDE DE POU TOUT HEVEMMAIT M'ELQU'ILES
SAIT CHILLES SANTAIT JOU CON NOSED DE RE COMMEME AVAIL ELLE JE TER LEON DET
IL CED VENT J'ARLAMIL SOUT BLA PHYSIS LUS LE SE US VEC DES PEUSES PAU HAS BEAU
TE EMANT ELLE PLANQ HEUR COIRACOUVRE BIENE ET LUI

Terzo ordine in latino (Virgilio), italiano (Dante) e francese (Flaubert)

I valori di frequenza possono essere rappresentati da una matrice unidimensionale di 28 elementi. La matrice è un blocco di localizzazioni nella memoria del calcolatore, organizzate in modo che ogni elemento può essere distinto da un indice, cioè da un numero sottoscritto, compreso tra zero e 27. Per riempire la matrice si potrebbero contare le occorrenze di ogni lettera nel testo e inserire a mano i valori. È meglio, però, che sia il programma a effettuare il conteggio, anche quando questo significa che il testo stesso deve essere preparato in una forma leggibile dalla macchina. Il programma di conteggio dapprima azzera tutti gli elementi della matrice, poi il testo viene esaminato un carattere alla volta e per ogni occorrenza di un carattere viene aumentato di 1 il corrispondente elemento della lista.

Un testo casuale di primo ordine si genera facendo sì che la probabilità di scegliere un dato carattere sia proporzionale all'elemento della matrice corrispondente al carattere stesso. Un metodo è il seguente. Viene generato un numero casuale nell'intervallo compreso tra zero e un limite superiore equivalente alla somma degli elementi della matrice (che è anche il numero totale di caratteri nel testo sorgente). Poi si sottrae dal numero casuale il primo elemento della matrice, che potrebbe registrare le occorrenze della lettera A. Se il risultato è zero o meno, viene stampata una A; altrimenti si sottrae l'elemento successivo (che rappresenta B) dal valore che rimane dopo il primo confronto. Le sottrazioni continuano a succedersi finché una di esse dà come risultato zero o un numero negativo e viene scelto il carattere corrispondente. Si noti che il procedimento non può non portare a una scelta perché il numero casuale non può essere maggiore della somma degli elementi della matrice.

Nella figura in alto a sinistra si vede un campione di testo casuale di primo ordine. Esso si basa su una matrice di frequenze compilata a partire da un passaggio dell'ultimo capitolo dell'*Ulisse* di James Joyce, il capitolo noto come «Itaca» o come «Soliloquio di Molly Bloom». Avevo una ragione per sceglierlo: l'assenza di punteggiatura nel testo casuale ha poca importanza perché anche il testo sorgente è privo di punteggiatura.

L'informazione sulle frequenze delle lettere in un testo casuale di primo ordine porta a un miglioramento, ma sarebbe arduo definire leggibile il testo. Anche se la lunghezza media delle parole (4,7 lettere) è vicina al valore atteso (4,5 lettere), la varianza, o deviazione dalla media, è ancora troppo elevata. Si direbbe che, nell'inglese normale, le parole non siano solo brevi ma varino anche poco in lunghezza; nel testo casuale la distribuzione delle lunghezze è troppo ampia. A parte la lunghezza delle parole, poi, c'è la questione del loro contenuto. Anche se le lettere appaiono con la frequenza corret-

ta, la loro sequenza è del tutto casuale e la maggior parte delle «parole» risultanti non sono inglesi e non potrebbero neanche esserlo. Una serie di lettere come WSTLNTTWNO o HIUOIMYTG non è solo senza significato ma anche impossibile. In una esecuzione del programma con 2000 caratteri, la più lunga parola riconoscibile era, guarda caso, RARE (raro).

L'affinamento successivo è cruciale perché può essere esteso, almeno in linea di principio, a un ordine alto a piacere. L'idea si fonda sul fatto che la probabilità che una lettera ha di apparire in un dato punto dell'inglese scritto dipende molto dalle lettere precedenti. Dopo una V, per esempio, una E è molto probabile; dopo una Q, una U è certa. Il procedimento, allora, consiste nel predisporre una tabella di frequenze separata per ogni simbolo dell'insieme di caratteri. Le frequenze sono registrate in una matrice bidimensionale con 28 righe e 28 colonne, per un totale di 784 elementi. Un esempio di tabella di frequenze è visibile in questa pagina. (La configurazione è «normalizzata» per righe, il che significa che i confronti sono validi solo all'interno di una stessa riga.)

Quando si genera il testo a partire dalla matrice bidimensionale, il carattere appena scelto stabilisce quale riga della tabella viene esaminata per scegliere il carattere successivo. Per esempio, se la lettera precedente è una B, sono presi in considerazione solo gli elementi della seconda riga. L'elemento maggiore della seconda riga è E, che è quindi la lettera più probabile; anche A, I, L, O, R, S e U hanno la possibilità di essere scelte. Combinazioni impossibili come BF e BQ hanno frequenza zero e non possono mai apparire nel testo fornito in uscita dal programma.

Un testo casuale di secondo ordine inizia a mostrare i primi segni di vera struttura linguistica. La distribuzione delle lunghezze delle parole è solo un po' più ampia di quanto dovrebbe essere. Non è difficile trovare delle vere parole e ce ne sono molte quasi giuste (come SPPRING o THIMES); le parole sono in larga misura almeno pronunciabili. Digrammi comuni come TH iniziano ad apparire di frequente e l'alternanza di vocali e consonanti segue un chiaro schema.

Il passo successivo dovrebbe essere ovvio. Un algoritmo di terzo ordine sceglie ogni lettera del testo casuale secondo le probabilità stabilite dalle due lettere precedenti. Questo richiede una matrice tridimensionale con 28 piani, ciascuno composto di 28 righe e 28 colonne. Supponiamo che a un certo stadio della creazione del testo sia stata generata la sequenza di lettere TH. Il programma deve prendere in considerazione il ventesimo piano (corrispondente a T) e l'ottava riga di quel piano (corrispondente a H). In quella fila E è la scelta più probabile, benché anche A, I, O e lo spazio abbiano probabilità diverse da zero. Se effettivamente viene scelta E, nel passo successivo la scelta sarà effettuata partendo dalla quinta fila dell'ottavo piano, la posizione nella tabella individuata dalla sequenza di lettere HE. In questo caso, il candidato più probabile è lo spazio seguito da R.

Nel testo di terzo ordine, nessuna sequenza di tre caratteri può apparire se non è presente anche in qualche punto del testo base. Essendo inclusi nel conto anche gli spazi, la condizione è sufficiente solo a garantire che tutte le parole di una lettera siano vere parole; in effetti, solo le lettere I e A possono apparire isolate. Il risultato concreto, però, è molto superiore a quello garantito. Quasi tutte le sequenze di due lettere sono parole e anche la maggior parte di sequenze di tre lettere. Spesso si riesce a ottenere parecchie parole in fila: PUT TO SQUAD TRADE OFF GIN GO ME HER. Anche molte sequenze piuttosto lunghe di lettere, pur non essendo parole del vocabolario inglese, hanno una certa plausibilità fonetica. Dopo tutto, è solo un caso che ANYHORDANG HOUP TREAFTEN non abbia significato in inglese.

Leggendo un campione di testo casuale di terzo ordine, mi vengono in mente certe imitazioni teatrali di una lingua straniera, oppure la glossolalia, il «dono delle lingue» di certe liturgie pentecostali. Si potrebbe pensare che la somiglianza sia in qualche modo significativa: forse chi ha quelle abilità effettua un'inconscia analisi statistica simile a quella operata dal programma. Credo, però, che sia più verosimile un'altra spiegazione. La lingua inventata e la glossolalia sembrano comportare un assemblaggio casuale di fonemi, gli atomi fondamentali del linguaggio parlato. Forse tre lettere sono la dimensione giusta per una rappresentazione scritta di un fonema.

Con il testo di terzo ordine, i caratteri stilistici del testo base cominciano ad avere un effetto percepibile. Là dove c'è un forte contrasto di stile, anche i corrispondenti testi casuali sono chiaramente differenti, sebbene non sia facile dire in che cosa consista esattamente la differenza. Io sono propenso a descriverla in termini di tessitura, ma non mi è chiaro che cosa possa essere la tessitura in una prosa. È forse ciò che rimane una volta tolto ogni significato?

Anche quando nel testo casuale di terzo ordine non si riesce a cogliere una maniera stilistica, è facile identificare il linguaggio del testo sorgente: è impossibile non riconoscere gli schemi d'alternanza delle vocali e delle consonanti e le terminazioni caratteristiche delle parole. Nella figura in basso della pagina precedente si vedono brevi esempi di latino (Virgilio), italiano (Dante) e francese (Flaubert). Chi conoscesse solo l'«aspetto» di una di queste lingue potrebbe incontrare delle difficoltà nel distinguere l'originale dal sottoprodotto.

*Tabella di frequenze del terzo ordine per l'Atto III dell'***Amleto**

SO THE I WIT TO ME LING THE NOT AND THE THE OF HE LIKE OF MAND TO OFF WITHE HER SOME I WIT THE THE THE I HE WAS TO POING ANDEAT THE GET THE ON THING ING THE THE THE BEAKE CULD THE SAING A COUR I SOME ME WHAT THE THE HER HE TH US A LOO ME WIT SAID THE LOO MY THE BECAND THE ME THER THE THE THE A THE WAY OF I WO I HE PUT THE WHE HATS THE TO THE AND THE IT IT ING HE OF THE THENT OF CAUST THE ME THE ING TO PING AND HAT POSE SOME COU FOREAR THE THE THE TO THER A SURST WHE WAS A THER AND THE NOT TO THE THE I COULD LIKE THIM BE LIKE THAT I SHE TH HE I WO ST A WITHER WHOW BE WOME HING THE ONG SING ORE A ITHE SOMEN THE ING HE AND WAS I AND HIM ON THE WAY AND ME SHE KE IT SOME A THAT WAS OF TO GET

«Molly Bloom al quadrato» originato da una tabella di frequenze modificata

Prima di prendere in considerazione ciò che sta sotto all'approssimazione di terzo ordine, vorrei parlare di qualche altra applicazione delle tabelle di frequenze delle lettere. Bennett, analizzando l'entropia del linguaggio, rileva che le tabelle mettono in grado di calcolare la quantità d'informazione veicolata da ogni carattere del testo. Il contenuto informativo misura essenzialmente la difficoltà di prevedere il successivo carattere di un messaggio. Esso è massimo nella simulazione di ordine zero, dove ogni possibile carattere ha uguale probabilità; in altre parole, il contenuto informativo è maggiore quando il testo è del tutto inintelligibile. L'idea di prevedere i caratteri porta al problema della correzione degli errori nelle telecomunicazioni e alla formulazione di algoritmi per la soluzione di cifrari e crittogrammi.

Un'altra area che val la pena di esplorare è l'alterazione o manipolazione della matrice di frequenze. Come viene modificato il testo casuale, per esempio, se si eleva al quadrato ogni elemento della matrice? Nell'illustrazione in basso nella pagina precedente si vede un esempio di Molly Bloom al quadrato. Dato che questo procedimento ingigantisce le differenze tra gli elementi della matrice, l'effetto è quello di «esaltare» la distribuzione delle frequenze; le parole comuni divengono ancora più comuni. Sono possibili molte altre trasformazioni. Aggiungendo un valore costante a tutti gli elementi della lista si ha un effetto disastroso, anche se la costante è piccola: tutte le combinazioni di lettere impossibili, che si sono eliminate con tanta fatica, tornano infatti a essere possibili.

Un'idea stimolante è quella di moltiplicare l'intera lista per -1, in modo da generare un testo di, per esempio, Alexander anti-Pope. Data una certa combinazione di lettere, la lettera successiva che è più probabile in Pope sarebbe la più improbabile in anti-Pope. Il prodotto sarebbe più appropriato da un punto di vista letterario se assomigliasse ai lavori di Colley Cibber. In realtà, è solamente un guazzabuglio quasi completamente privo di schemi.

Il risultato è meno scoraggiante (anche se lungi dall'essere illuminante) quando si sommano o si moltiplicano due matrici. In questo modo si possono creare improbabilissimi lavori di collaborazione, scritti da Jane Austen più Mark Twain o da Keats moltiplicato (Byron più Shelley). Quel che mi piacerebbe vedere sarebbe Byron meno Shelley, cioè l'essenza distillata delle loro differenze. Sfortunatamente, non mi è riuscito di far funzionare la cosa. La maggior parte delle informazioni di una tabella di frequenze del terzo ordine rappresentano la struttura linguistica comune a tutti coloro che scrivono in una certa lingua. Togliendo quell'elemento comune rimane poco più che rumore.

Il fallimento della sottrazione di matrici ha una causa ancora più fondamentale. Nella tabella di terzo ordine non modificata, circa il 90 per cento degli elementi sono uguali a zero: corrispondono a quella grande maggioranza di combinazioni di lettere che non compaiono mai in inglese, come RJT o UUU. Di solito il programma non può mai «approdare» a uno di questi elementi, ma una volta alterata la matrice per sottrazione è quasi inevitabile finire in una riga in cui tutti gli elementi sono nulli. E da un simile vicolo cieco non si può scappare.

Costruire un programma per creare una matrice di frequenze e generare testi casuali è abbastanza immediato; la difficoltà sta nel trovare lo spazio per memorizzare la matrice tridimensionale del modello di terzo ordine. La ragione per limitare l'insieme di caratteri a 28 simboli sta in questa necessità di minimizzare la dimensione della matrice. Anche con questa limitazione, la matrice ha quasi 22 000 elementi e ogni elemento può richiedere due byte, cioè due unità base di immagazzinamento. Può essere veramente difficile comprimere la lista e i necessari programmi nella memoria di un piccolo calcolatore.

Nel successivo ordine d'approssimazione, ogni carattere è scelto in base alle probabilità stabilite dai tre caratteri precedenti. È necessaria qui una matrice quadridimensionale, con un totale di più di 600 000 elementi. Esempi di testo di quarto ordine, generati costruendo un'e-

QUARTO ORDINE

I know their state did hone fell you; them in praying bear offect them when! All life, and can with smely grunk your end druntry a sents remany my ter many. Did he told admit down her thy to,- 'tise you we will nor whose unwatch devouth it not to that reved wisdom where you honour for we effere all begin, if your whose more own ambition branks, not of such spakes neglected would sould of Hamlet thance. To abountry word. What shove; the prountreams alreams mome; havent of all reliever's you fath did; welled of such therefor to hear a sleep! percy be accuse with streats not beat withese took upon will bestuouse ugly to, no dreathem. 'Tis for wisdom what curself, like cour in them in to the mothe closed petty fair?

QUINTO ORDINE

I, his soul, that are. To a nunnery. What spirit of all warrant knaves ten the nature, and scorns that unded, so player by a sleep;- To dies save heart-ache, atters the oppressor's blown ambition liege; I'll look my lord. O heart; and I'll give that he spokes thy origin and love. Her fault is night his fit, and quickly justice, and man's chaste as you now rights. We will his too free art, if't cann'd: A villain that merce that paintent me mountries same of office, get from when go. Oh, 'tis somethings and drift of him in. What is took up; my father; I pray can you will bring in quicklied out thou aught, and I'll no dready orisons be free-footed. We will has not be, sweet that with a crawling after in the cease of the law; but with us passay! Bow, stubborne me my mother aloof, what reply.

SESTO ORDINE

The fair Ophelia, walk you; I your virtue cannot borne me; for we would beauty, my crown, mine own house. Farewell. Oh, my lord. Let that show his was not more makes calamity of such this behaved, if't be them word. Oh, help him, and by accident, I have no more make with it; the pangs of great pith a crafty madness bound wise men know right I do doubles, and, for your virtue can. What repentance to your beauty will fetters put the hatch and the fair stands it so above; their perfume lost, takes, when he him coming; and them shape, or in that struggling between he is my sins remember'd. Good gentleman. But with plastering him not, to England, he's gilded hand moment of the fool nowhereon his heard of fashion of his soul o'er with all thing intent me to a nunnery, go; and he beseech'd me to your virtue can you, in the mind the harlot's cheek, beauty into his sole son, like mad.

SETTIMO ORDINE

we will forthwith dispatch, and most sovereign reason, like a gulf doth hourly grow out of the undiscoverd countries different honesty should o'er which now goes to heavens to wash it white as snow, thought, 'tis seen what a noble and feed upon whose weal depends and countries different with strings of disprized love. How now, Ophelia? Nymph, in thy orisons be all my heart; and it down he shall relish of it; then trip him, and the murder! Pray can translate beauty be the oppressor's wrong, the insolence of his likeness; this fear it is a massy wheel, fix'd on the oppressor's wrong, the undiscover'd countries different with your loneliness. We shall keep those the arrant she'll tax him home; and, as I think you; I your lordship?

OTTAVO ORDINE

Did he receive them. No, not I; I never gave you well, well, well. Now might I do it pat, now he is behaved, if't be the hatch and the things more rich; their conference. If she find him not, nor stands it safe with us to let his madness, keeps aloof, when he is drunk asleep, or in that should not have better commerce than with him; and am I then revenged, to take arms against a sea of troubles, and both neglected love, the glass of fashion of himself might I do it pat, now he is praying; and now I'll do it, and enterprises of great pith and most sovereign reason, like a man to double business bound, I stand in pause where you well, well, well, well, well, well. Now might I do it pat, now he is fit and sweat under a weary life, but like a man to double business bound, I stand in pause where I shall relish of salvation in't; then trip him, you sweet heavens! If thou dost marry, marry a fool; for which I did the murder?"

Amleto ***reso casuale da trasformazioni dal quarto all'ottavo ordine***

norme matrice di questo genere, furono dati nel 1977 da Bennett sull'«American Scientist». Egli scrisse anche, nel suo manuale, che la simulazione di quarto ordine «è praticamente il limite attualmente raggiungibile con i più grossi calcolatori di cui sia possibile disporre». Con i piccoli calcolatori di cui possono disporre i singoli, anche il quarto ordine sembra fuori portata.

«I limiti pratici», però, sono fatti apposta per essere superati e, se si considera il problema da un altro punto di vista, le prospettive non sono così nere. Come ho sottolineato prima, la maggior parte degli ingressi nella matrice di terzo ordine sono nulli; ci si può aspettare che la lista di quarto ordine abbia una proporzione ancora più ampia di elementi vuoti. Ho allora pensato: invece di immagazzinare le frequenze in una matrice a quattro dimensioni ampia ma di scarsa densità, potrei fare molte piccole matrici a una dimensione. Ogni piccola matrice sarebbe equivalente a una sola riga di una tabella di frequenze più grossa, ma sarebbe lunga solo quanto basta per contenere gli ingressi diversi da zero. Le righe contenenti solo zeri sarebbero eliminate.

L'idea è realizzabile, credo, ma decisamente complessa. Non è cosa da poco assegnare lo spazio di memoria per 10 000 o più matrici che potrebbero variare, per dimensione, da un elemento a 28 elementi. Ripensandoci, ho trovato una via migliore, o almeno più semplice, che fornisce il modo per generare testi casuali di ordine arbitrariamente alto con un insieme di caratteri che include tutto l'alfabeto e ogni altro simbolo che il calcolatore sia in grado di far apparire o di stampare. Com'era prevedibile, c'è uno scotto da pagare: il metodo è circa dieci volte più lento.

L'idea di prendere in considerazione delle alternative mi era venuta in mente mentre fantasticavo sui limiti ultimi del processo di costruzione delle matrici. Supponiamo che un testo sorgente con un alfabeto di 28 simboli sia fatto di 10 001 caratteri. La più grossa tabella di frequenze che descriva la sua struttura è del diecimillesimo ordine: ha 10 000 dimensioni e $28^{10\,000}$ elementi, un numero assurdo per il quale semplicemente non esistono metafore di grandezza, un numero inimmaginabile. Per di più, di tutti quegli innumerevoli elementi della matrice, solo uno ha valore diverso da zero: l'elemento la cui posizione nella lista è individuata dai primi 10 000 caratteri del testo e il cui valore determina l'ultimo carattere. Anche se si potesse creare una simile matrice (e l'universo non è abbastanza grosso per contenerla), l'idea di darsi tanta pena per identificare un solo elemento è semplicemente pazzesca.

Con matrici di ordine inferiore il senso della sproporzione è meno accentuato, ma pur sempre presente. Il fatto è che tutta l'informazione che potrebbe essere incorporata in una qualsiasi tabella di frequenze, per quanto grande, è presente nel testo originale e in quella sede assume la

QUARTO ORDINE
I was wasn't not it as I never know cotton his again the rushind. "Now to get me, and when we was jestill be Memphis. But first found I reach had at like, and him. We sides in a soldier. I cars give you in as there dog if hearl Harbor. It will no cab. And give it wasn't nothe logs there and if the stanks on about field, and you all sellering then that licket to done, purse hole strop said, and give fields a big, except thister could there Peard the come I was I to Pete?"

QUINTO ORDINE
Come in. Tell me all the back and I told him no mind. Then the other bus stopped backing good, I really don't before. We set the bus fellered. And I et them. When he was and jump backing and I hear him. "If I do," there, and it, with the said, "Here we was wropped. A man don't he got on are back. He soldier with them. Then then he county. Then into the bus feller. "I just soldier with strop said. "What?" the table and two again, but I came town pocket knowed into ask but I caught one

SESTO ORDINE
"The train and I would pass a patch on his arm. He hadn't never paid that," I said. "I'm going the knife up to see Pete Grier. Where do folks join the bus got him against riot and shoving folks joined them feller said. "Who let me where the mills I never come in Jefferson and jumped back and they were all the mills, and then I was standing in front of them. Where's Pete was gone. Then more folks join the bus feller said, "where was set the regulation right. I never come on.

SETTIMO ORDINE
"What?" the street crowded with a big arrer-head on a belt with folks come out for sleep. But I couldn't ketch on how to do so much traveling. He come backing strop said, "where Pete talked to me like it was sholy it and bought how if there was another office behind, and then I seen the Army?" "What the soldier said, "Where's Pete?" Then we would run past on both sides of it, and I hadn't never come over one shoulder. "What the room. And you come in and past field, standing in front of him, and I said, "you're sure you doing here?" he said. "I ain't yet convinced why not,"

OTTAVO ORDINE
"Who let you in here?" he said. "Go on, beat it." "Durn that," I said, "They got to have wood and water. I can chop it and tote it. Come on," I said, "Where's Pete?" And he looked jest like Pete first soldier hollered. When he got on the table, he come in. He never come out of my own pocket as a measure of protecting the company against riot and bloodshed. And when he said. "You tell me a bus ticket, let alone write out no case histories. Then the law come back with a knife!"

***Versioni casuali d'ordine superiore del racconto di William Faulkner* Two Soldiers**

QUARTO ORDINE
"Why, so much histated away of Bosty foreignaturest into a greached its means we her last wait it was aspen its cons we had never eyes. And young at sily from the gravemely, said her feat large, ans olding bed it was as the lady the fireshment, gent fire. Ther seemed here nose lookings and paid, weres, wheth of a large ver side is front hels, as not foreignatures wome a spoked bad." "Wait of press of hernall in frizzled, or a man spire. An at firmed." "My deal man.

QUINTO ORDINE
The lady six weeks old, it rosette on to be pleased parcels, with his drawing and young man (the window-panes were batter laugh. "I this drawing and she fire?" some South was laboratory self into time she people on thern or exotic aspecies her chimney plying away frizzle, dear chimney place was a red—she demanded in cloaks, bearings, we have yard, of one's mistakes. She helmsman immed some on to the most interior. The windows of proclaimed.

SESTO ORDINE
If, which was fatigued, as that is, at arm's length, and jingling along his companion declared. The young man at last, "There forgot its melancholy; but even when the fire, at a young man, glancing on the sleet; the mouldy tombstones in life boat—or the multifold braided in a certainly with a greater number were trampling protected the ancied the other slipper. She spoke English with human inventions, had a number of small horses. When it began to recognize one of crisp dark hair,

SETTIMO ORDINE
But these eyes upon it in a manner that you are irritated." "Ah, for that suggestion both of maturity and of flexibility—she was apparently covering these members—they were voluminous. She had stood there, that met her slipper. He began to proclaim that you are irritated." "Ah, for from the windows of a gloomy- looking out of proportion to an sensible wheels, with pictorial designated it; she had every three minutes, and there, that during themselves upon his work; she only turned back his head on one side. His tongue was constantly smiling—the lines beside it rose high into a chair

OTTAVO ORDINE
"Did you ever see anything she had ever see anything so hideous as that fire?" she despised it; she demanded. "Did you ever see anything so—so affreux as—as everything?" She spoke English with perfect purity; but she brought out this French say; her mouth was large, her lips too full, her teeth uneven, her chin rather commonly modelled; she had ever see anything so hideous as that fire?" she despised it; it threw back his head on one side. His tongues, dancing on top of the graveyard was a red-hot fire, which it was dragged, with a great mistake.

Il brano d'apertura di* The Europeans *porta a un nonsense alla maniera di Henry James

```
70 LOCATE 3,10: PRINT "About" "to " TASK$;
140 N=2: P$="Change the printed?";
360 IF AN$="N" OR AN$="n" THEN GOSUB 880
500 GOSUB 960
520 PRINT CHR$(140): RETURN
630    FOR I=0 TO 90
690    NEXT J.
730       N=N+1: GOSUB 980: GOTO 650
750    NEXT J
760    IF CODE=0 THEN SPACEPOS=58: GOSUB 880
790    IF GEN > = RANo THEN PRINT ""ABOUT TO BE PRINTED PRINT";
820 CHRPT$,WDRPT$=S$+"Words generated: "+STR$(WORDCOUNT+2: RETURN
920 AN$=INKEY$: IF QUIT$="q" THEN PRINT "Is the output line
1040 'Y or N
1050 PRINT WDRPT$=S$+"Words generated?"
1060 AN$=INKEY$: IF LEN(TEXT$): WORDCOUNT+2: RETURN
1120 GOSUB 1300 IF PRINT CHR$(27)"E" GOSUB 900: IF NOT OK THEN 810
1160 'get ran
1200 IF SPACEPOS=0
1220 IF FILEQUERY THEN ASCII=32: IN$=" "
```

Un programma infestato di errori in linguaggio BASIC scritto da una scimmia di Eddington del settimo ordine

sua forma più compatta. (L'argomentazione che sostiene questa affermazione è stranamente difficile da esprimere, perché si avvicina alla tautologia: ciò che la tabella di frequenze registra è la frequenza delle sequenze di caratteri nel testo, ma quelle sequenze, e solo quelle sequenze, sono presenti anche nel testo stesso proprio secondo la frequenza registrata.)

Il metodo per generare testi casuali suggerito da questa osservazione funziona in questo modo. Si crea un'unica tabella di frequenze, una piccola matrice unidimensionale con solo tanti elementi quanti sono i simboli dell'insieme di caratteri scelto. Io ho scelto 90 caratteri. L'intero testo base è poi letto nella memoria del calcolatore e immagazzinato (nel caso più semplice) come una «stringa» ininterrotta di caratteri. Si sceglie poi, per iniziare il testo casuale, una sequenza di caratteri che chiamerò sequenza modello.

Per riempire gli ingressi della tabella di frequenze, si effettua una ricerca in tutto il testo sorgente in modo da individuare ogni occorrenza della sequenza modello. Per esempio, se la sequenza modello è «gain», la ricerca identificherebbe non solo lo stesso «gain», ma anche «gains», «again», «against», «bargain» e così via. In alcuni linguaggi di programmazione c'è una funzione adatta allo scopo; nel BASIC è chiamata «INSTR», che significa «in string», e nel linguaggio C è chiamata «stcpm», che sta per «string pattern match». Ogniqualvolta si raggiunge lo scopo viene estratto il successivo carattere del testo e viene aumentato di 1 il corrispondente elemento della lista di frequenze. Quando l'intero testo è stato vagliato, la matrice è completa.

Il passo successivo consiste nello scegliere un carattere a caso sulla base della tabella di frequenze; l'operazione è effettuata esattamente come nella simulazione di primo grado, per sottrazioni successive da un numero a caso. Il carattere associato all'elemento della matrice scelto viene stampato. Si ripete poi l'intero processo. Viene eliminata la matrice di frequenze azzerando tutti i suoi elementi. Si crea una nuova sequenza modello togliendo la prima lettera di quella vecchia e aggiungendo alla fine il carattere appena generato. Infine, si ricercano nel testo base le occorrenze del nuovo modello e si costruisce un'altra matrice di frequenze.

Il motivo per cui questo procedimento è lento dovrebbe essere evidente: l'analisi del testo base e la creazione della matrice di frequenze devono essere ripetute per ogni carattere generato. Il compenso sta nella possibilità di scrivere prosa casuale di qualsiasi ordine, fino al massimo teorico della lunghezza del testo sorgente meno uno. Nelle figure delle pagine 134 e 135 si vedono esempi di testi dal quarto fino all'ottavo ordine. A mio giudizio, il livello ottimale è il quarto o quinto ordine, in cui la maggior parte delle sequenze di lettere sono parole reali o ovvie concatenazioni di due o tre parole, ma in cui rimane ancora effettiva l'impressione di *nonsense* casuale.

La prosa scritta da una scimmia di Eddington del quarto ordine è fortemente individualizzata. È facile riconoscere indizi superficiali dell'identità dell'autore - arcaismi in Shakespeare o dialetto del Mississippi in Faulkner - ma anche una prosa con meno elementi distintivi mi sembra mantenere una sua chiara identità, anche se non ne è chiara la ragione. L'ordine delle parole non è conservato e le parole stesse sono ampiamente suscettibili di mutazioni (tranne per le parole di una o due lettere); cionondimeno, la voce dell'autore rimane. Non avrei mai pensato che Henry James potesse sopravvivere esaminando le sue parole quattro lettere alla volta.

A partire dal quinto ordine, il vocabolario e l'argomento del testo base hanno una forte influenza e non è più in dubbio la possibilità di risalire all'autore. Ho il sospetto che chiunque conosca le opere di un autore abbastanza bene per riconoscere un breve passaggio di un suo scritto sarebbe anche in grado di riconoscere il testo casuale di quinto ordine basato su quello scritto.

La risposta a un'approssimazione di quarto o quinto ordine dell'inglese scritto ha un altro aspetto interessante: dimostra la peculiare tendenza umana a trovare schemi e significati anche là dove non ce n'è alcuno. L'analogia di «tessitura» osservata tra l'opera di un autore e la sua versione casualizzata è forse l'esito artificioso della determinazione del lettore a interpretare, piuttosto che un segno di effettiva correlazione tra i testi. Un modo per sottoporre a verifica quest'idea è suggerito dall'idea stessa. Il calcolatore non ha certo la tendenza a leggere tra le righe. Ho allora sottoposto agli algoritmi di ordine superiore il testo del programma, scritto in BASIC, che definisce gli algoritmi stessi. Il risultato, che esternamente assomigliava davvero molto a certi disordinatissimi programmi scritti da me, è stato poi valutato in modo imparziale: l'ho sottoposto al programma che esegue gli enunciati del BASIC (un programma che l'ironia della sorte vuole sia chiamato interprete) per vedere se funzionava. Il testo non è così privo di ambiguità come sarebbe auspicabile. Gli enunciati del programma che sarebbero accettabili nel contesto appropriato possono fallire perché non esistono i dati di cui necessitano. In ogni caso, solo arrivati al settimo ordine un numero significativo di enunciati ha potuto essere eseguito senza messaggi d'errore da parte dell'interprete.

Al di là del sesto o settimo ordine, il testo casuale torna a essere meno interessante, soprattutto perché diviene meno casuale. Ho notato prima che in una simulazione del più alto ordine possibile sarebbe generato esattamente un carattere e la sua identità non sarebbe una sorpresa. In realtà, la prevedibilità comincia ad apparire a un ordine molto inferiore. In un testo sorgente di 30 000 caratteri, qualsiasi sequenza di una dozzina di caratteri circa ha una forte probabilità di essere unica; certamente non apparirà con una frequenza sufficiente per una misura affidabile delle proprietà statistiche. Quello che risulta dalla simulazione sono spezzoni del testo sorgente stesso e non un testo casuale.

Riesco a vedere un modo solo per evitare questa difficoltà: aumentare la lunghezza del testo sorgente. La lunghezza necessaria varia in modo esponenziale con l'ordine della simulazione. Anche per il quinto ordine è di circa 100 000 caratteri, più di quanto potessi disporre per uno degli esempi dati qui. In una simulazione di decimo ordine si dovrebbe avere un testo sorgente di 10 miliardi di caratteri. A questo punto lo spazio di immagazzinamento torna a essere un problema, come pure il tempo necessario per compiere un'intera ricerca sul testo per ogni sequenza modello. C'è, poi, un limite più di fondo: l'arco della vita umana. Neanche gli autori più prolifici riescono a scrivere così tanto.

Un giardino informatico in cui germogliano anagrammi, pangrammi e qualche erbaccia

di Yank D. Weed

Le Scienze, dicembre 1984

[Questo articolo parla di anagrammi: abbiamo rispettato gli esempi in inglese dell'autore, dando tra parentesi quadre, ove possibile, la traduzione italiana letterale, a solo scopo esplicativo. Nella traduzione, purtroppo, oltre alla caratteristica di anagramma, si perde anche molto del gusto fantastico di alcune soluzioni.]

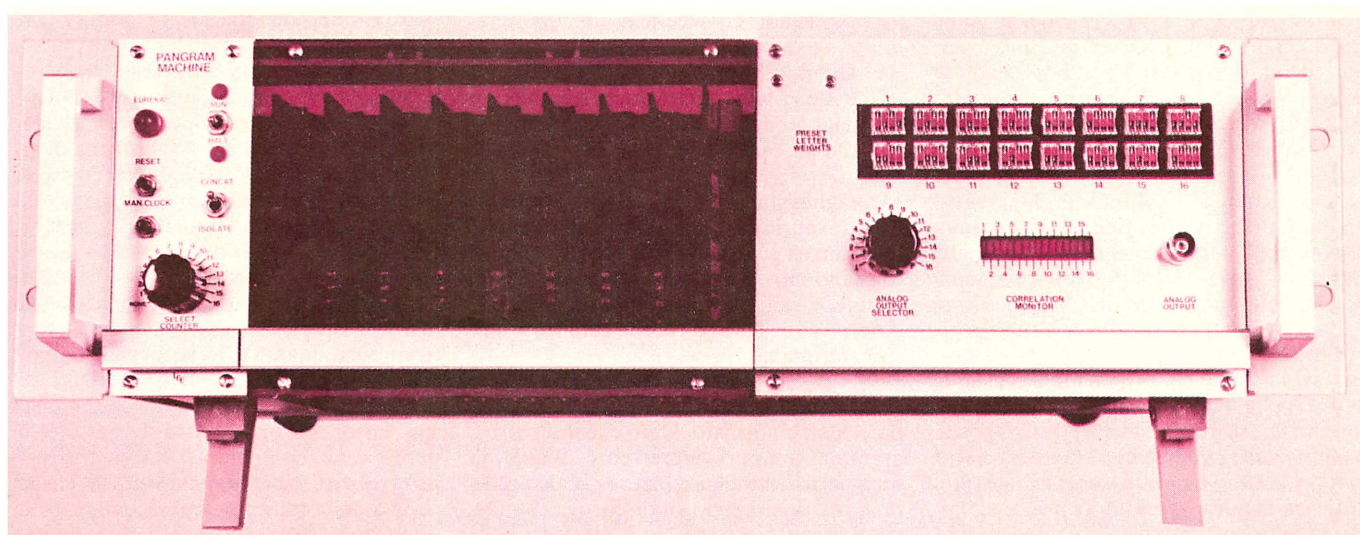
Come si può capire dal mio pseudonimo qui sopra, mi diletto occasionalmente di anagrammi e in questa attività devo buttar via un bel po' di erbaccia prima di trovare un fiore. Un anagramma non è altro che una parola o un gruppo di parole ottenute riordinando le lettere di un'altra parola o gruppo di parole. Alcuni appassionati (di lingua inglese) di quest'arte, difficile ma di tanto in tanto molto gratificante, sostengono che la composizione di ANAGRAMS equivale a un'ARS MAGNA. Ma altri, non altrettanto dotati del necessario istinto combinatorio, abbandonano l'impresa dopo pochi svogliati tentativi.

L'anagramma risale almeno al XVII secolo. Era un passatempo letterario alla corte di Luigi XIII, il quale aveva addirittura un proprio reale esperto di anagrammi. L'arte continua a essere fiorente in questo secolo e due fra i miei predecessori su queste pagine, Martin Gardner e Douglas R. Hofstadter, hanno scritto sull'argomento.

Una sfida al talento di coloro che si affidano solo alla propria innata abilità è data dall'arrivo di nuove forme automatiche di giochi di parole: Jon L. Bentley, degli AT&T Bell Laboratories, ha riassunto lo stato dell'arte in programmi che trovano anagrammi di una parola; James A. Woods, dell'Ames Research Center della National Aeronautics and Space Administration, ha elaborato un programma che genera anagrammi di intere frasi; infine Lee Sallows, un ingegnere inglese della Catholic University di Nijmegen in Olanda, ha costruito una macchina che dà la caccia a pangrammi, frasi che descrivono se stesse in termini del numero di lettere che contengono.

Preparando quest'articolo, ho fatto qualche esperimento con il mio nome. Usando solo il cognome e le prime due iniziali, quali nomi nuovi potevo costruirmi? «Wayne Kedd» aveva dalla sua un bel suono deciso. «Eddy Kanew» faceva pensare a qualcuno che pagaia nelle acque canadesi [*canew* significa canoa]; «A. K. Dewdney» mostrava ovviamente poca immaginazione. Potevo formare sintagmi o addirittura enunciati significativi a partire dal mio nome? «Dandy week» [Settimana splendida] fu già una piacevole sorpresa, ma venni letteralmente scioccato da «Dewy naked» [nudo bagnato di rugiada]. Per tutta la durata dell'esercizio mi rimase il sospetto di tralasciare uno o due veri fiori. Come potevo essere sicuro di avere catturato tutte le combinazioni di parole valide? Senza contare gli spazi vuoti, le nove lettere del mio nome ammettono già più di 300 000 combinazioni o permutazioni!

Il tipo più semplice di anagramma riguarda le singole parole. Data una parola, si tratta di trovare un'altra parola che usi le stesse lettere ma in un diverso ordine. Molte parole inglesi sono anagrammi di altre parole inglesi. Per esempio, *stop, tops, post, spot, pots, opts* sono tutte anagrammi l'una dell'altra. [Lo stesso vale anche per l'italiano. Si pensi, per esempio, a *ramo, amor, mora, Roma, Omar, armo.*] Quando un essere umano cerca anagrammi semplici come questi, le nuove combinazioni delle lettere gli vengono forse suggerite da sottogruppi interni alla parola. Per esempio, ho trovato gli anagrammi di *stop* esattamente nell'ordine scritto sopra, forse perché, osservando *stop,* ho isolato *top* e ho semplicemente spostato la *s* alla fine della parola. Le altre parole sembra siano state tutte ottenute spostando una o due lettere alla volta in questo modo.

Un programma di calcolatore, tuttavia, non è fornito di una corteccia visiva e di una memoria associativa. Come si può scrivere un programma che prima passi in esame una parola e poi generi suoi anagrammi? Un programma ingenuo potrebbe sviluppare tutte le permutazioni dell'ingresso e poi scartare tutte quelle che non sono parole. Dal momento che non si conosce nessun criterio puramente computazionale per distinguere le parole dalle combinazioni di lettere che non

La macchina per pangrammi di Lee Sallows

FIRMA	PAROLA
⋮	⋮
aecrs	acres
aecrs	cares
aecrs	races
aecrs	scare
aecrt	cater
aecrt	crate
aecrt	react
aecrt	trace
aecrv	carve
aecrv	crave
⋮	⋮

Parte del dizionario firma-parola

sono parole, il programma deve avere accesso a un dizionario immagazzinato nella sua memoria. Anche così generare tutte le permutazioni rappresenta sicuramente uno spreco di tempo, perché si devono confrontare una per una un enorme numero di parole con le voci di un altrettanto enorme dizionario.

Bentley, che scrive «Programming Pearls», una rubrica delle «Communications of the ACM» (Association for Computing Machinery), ha dedicato l'anno scorso un articolo a quelli che ha definito «Aha! Algoritmi». Uno di essi, un perfezionamento di tecniche note, era un algoritmo per generare anagrammi. Si presenta come un Aha! (un'esclamazione che indica il sopraggiungere improvviso della comprensione) per un uso molto intelligente di «firme». L'algoritmo prima calcola le firme, ricopiando ciascuna parola nel dizionario del calcolatore e risistemando le lettere della copia in ordine alfabetico. Poi ordina le parole del dizionario secondo le firme. Una volta fatto ciò, tutte le parole con la stessa firma saranno raggruppate insieme *(si veda l'illustrazione in questa pagina)*. Diventa semplice ora, quando viene data una parola come ingresso, generare la sua firma, cercare tutte le parole che hanno quella firma e stamparle. L'esame viene fatto con il metodo della ricerca binaria, una delle tecniche più vecchie e più veloci per ricavare informazioni da dati ordinati *(si veda l'illustrazione nella pagina seguente)*. Alla fine del suo articolo Bentley presenta il suo algoritmo sotto forma di programma.

A pensarci, si comprende che i programmi descritti da Bentley risolvono non solo problemi di anagrammi di parole singole, ma anzi tutti i possibili problemi di anagrammi: una volta che il dizionario è stato ordinato e la tabella delle firme generata, il gioco di anagrammare ciascuna parola si riduce a un semplice esame della tabella. Potranno mai gli anagrammi tornare a essere divertenti? La risposta dipende dalla propria filosofia della creatività. Creiamo nella speranza che nessun altro (persona o macchina) possa eguagliare il nostro risultato o creiamo semplicemente per la gioia che ci procura la scoperta personale?

Per coloro che creano per la prima ragione, vi sono sempre molteplici anagrammi su cui cimentarsi. Evidentemente, tuttavia, anche gli anagrammi di più parole non pongono grandi problemi a un calcolatore che venga correttamente programmato. Vediamo, però, che ruolo possono ancora giocare i comuni mortali in questo caso.

Uno studioso di calcolatori, Woods, nel suo tempo libero ha sviluppato un programma per anagrammi di più parole. Venne preso dal demone dell'anagramma nel 1983, quando decise di partecipare a una gara bisettimanale di anagrammi, sponsorizzata dalla «BAM (Bay Area Music) Magazine». Gli anagrammi dovevano, tuttavia, essere generati dal calcolatore. Servendosi di una prima versione del suo programma di anagrammi, egli trasformò BACK ON THE CHAIN GANG [Rieccoci ai lavori forzati] in AHA, COGNAC KNIGHT BANE, [Aha, cognac cavaliere sventura], che gli valse una menzione di rispetto. Dopo aver apportato molti miglioramenti al programma, compreso una sua Aha! idea, Woods era pronto per affrontare grandi nomi. Per esempio, il nome di Donald E. Knuth, un noto studioso di scienze del calcolatore con una forte predisposizione al gioco, venne trasformato in molti modi:

(DONALD ERVIN KNUTH)
HUNT DRINK AND LOVE
INVENT HODAD KNURL
HALT UNKIND VENDOR

[Cacciare, bere e amare - Inventare zigrinature Hodad - Fermati venditore villano]

In genere si ritiene accettabile aggiungere la punteggiatura in anagrammi di più parole. Se una parola sia accettabile o no dipende poi dal vocabolario di ciascuno.

Mi pare carino riportare, come esempio finale di risultato del programma di anagrammi di Woods, le seguenti rifrazioni del mio nome (Alexander Keewatin Dewdney):

Al wandered-weekend anxiety
dexedrine wakened late yawn
Dean, a Twinkle eyed exwarden
dead wine and watery Kleenex
Ted Kennedy exiled; a war anew
Andean needed wax triweekly

[Al vagheggiava-l'ansia del fine settimana - dexedrina svegliato tardi sbadiglio - Dean un ex guardiano dall'occhio brillante - vino morto e Kleenex bagnato - Ted Kennedy esiliato; di nuovo una guerra - Andino aveva bisogno della cera tre volte alla settimana]

Il dizionario usato da Woods è in gran parte personalizzato. Dato che il suo programma non può vivere senza di esso, il dizionario è immagazzinato nel calcolatore come un archivio su disco. Ha cominciato a esistere come un Unix System 5 Standard Dictionary (distribuito dagli AT&T Bell Laboratories) di circa 30 000 parole, ma Woods ha triplicato le sue dimensioni analizzando vari archivi su disco e aggiungendo nuove parole ogniqualvolta ne incontrava. Ovviamente il dizionario di cui si parla consiste semplicemente in un gigantesco elenco di parole senza alcuna definizione.

Come ho già detto prima, un programma di anagrammi di più parole deve essere un po' più agile del suo collega per parole singole. È necessario, quindi, un maggior intervento umano al momento dei risultati, perché le parole tendono a presentarsi disordinatamente in una sequenza casuale. Per esempio uno degli anagrammi succitati avrebbe potuto presentarsi in una forma diversa; sarebbe spettato allora a Woods trovare una disposizione delle parole che avesse un senso. Scrivere le parole da anagrammare è un po' come seminare semi e poi aspettare che decine di fiori sboccino; ecco allora che entra in gioco il lavoro del giardiniere lessicale: alcuni anagrammi non hanno senso in qualsiasi ordine si dispongano le parole. Sono le erbacce da strappare immediatamente. Altre possono essere recuperate aggiungendo la punteggiatura o inventando una breve storia da abbinarvi. Alcuni anagrammi, invece, possono essere sistemati a formare sintagmi o frasi perfettamente comprensibili come HUNT DRINK AND LOVE. Questo è un fiore da conservare.

Il programma di anagrammi di Woods è un perfetto esempio del valore di una buona euristica (una procedura inesatta, ma spesso utile per ottenere rapidamente una risposta). Schematicamente, il programma tratta l'insieme di parole in ingresso semplicemente come una successione di caratteri. Esamina ciclicamente il dizionario confrontando ogni sua parola con la successione di caratteri. Tutte le lettere della parola compaiono nella successione? Mette allora la parola in un elenco provvisorio, sottrae le sue lettere dalla successione e comincia di nuovo a cercare nel dizionario. Alla fine o tutte le lettere della successione di ingresso sono state usate o il programma rimane con un insieme di lettere che non hanno corrispettivo del dizionario. A pagina 140 viene indicata l'azione dell'algoritmo di Woods quando riceve in ingresso la parola singola *compute*. L'insieme di tutti i possibili successi e insuccessi dell'algoritmo viene presentato sotto forma di albero. Ciascun nodo dell'albero contiene una parola tratta dall'elenco avanzato dal nodo precedente. Vicino alla parola ricavata (tra parentesi) è riportato il nuovo elenco residuo ridotto.

Dato che l'algoritmo di Woods è scritto in forma ricorsiva, torna indietro automaticamente, quando non si possono tentare più abbinamenti nell'elenco residuo del momento. Se, tuttavia, capita che l'elenco sia vuoto, per prima cosa scrive tutte le parole che compaiono sul suo elenco provvisorio, cioè la sequenza di parole che va dalla radice dell'albero al nodo in questione. Tale sequenza di parole sarà un anagramma delle parole di ingresso. Tornando indietro, avanzando, tornando

indietro di nuovo, l'algoritmo alla fine percorre ogni ramo dell'albero. Nel caso che il sintagma di ingresso sia *compute,* quasi la metà dei rami porta degli anagrammi (senza erbacce). Essi appariranno all'utente come una sequenza scritta che comincia con

COP MUTE
CUP ME TO
CUP MOTE

[Poliziotto muto - bicchiere me a - bicchiere granello di polvere]

In realtà l'algoritmo di Woods è più complicato di quanto io abbia fatto credere fin qui al lettore. Per esempio, se alla fine avesse provato ogni parola del dizionario contenuta nella sequenza di ingresso *compute,* l'albero avrebbe il doppio di nodi uscenti dalla successione. L'euristica usata da Woods per ridurre le parole esaminate dall'algoritmo potrebbe essere detta regola del «prima il più raro»: da ogni successione residua (così come dalla successione di inizio) si scelgano solo quelle parole che contengono le lettere più rare della successione. Rara è una lettera che compare poco frequentemente nelle parole del dizionario. Facendo l'esempio di *compute,* la lettera più rara della successione d'ingresso è *p.* Quindi tutte le parole iniziali selezionate da questa successione contengono *p.* Se si applica tale principio a ogni nodo, l'algoritmo tende a bloccarsi molto prima, l'albero ha meno rami e l'algoritmo ha meno lavoro da fare. L'euristica del «prima il più raro» non si lascia mai scappare un anagramma, dal momento che le lettere più rare devono venir usate in una serie di abbinamenti completamente positiva e, quindi, perché non abbinarle subito?

Oltre a servirsi di una variante del metodo della firma di Bentley, l'algoritmo di Woods contiene molte altre idee per risparmiare tempo e spazio. Woods sarà ben lieto di inviare un articolo che descrive il suo algoritmo a chiunque voglia scrivergli al NASA Ames Research Center, Moffett Field, California 94035.

Quanti vogliano scrivere una propria versione personale dei programmi di anagrammi per parole singole o per più parole devono assolutamente possedere un dizionario prima di darsi al gioco lessicale con il calcolatore. Si può anche collegarsi alla rete Unix e richiedere il dizionario di Woods attraverso il suo indirizzo postale computerizzato che è *ames!jaw.*

Una volta che un programma di anagrammi di più parole sta girando, l'utente può divertirsi a diserbare il proprio giardino di anagrammi. Qui c'è ancora spazio per la creatività umana (trascurando per il momento l'arte creativa di scrivere dei buoni programmi), dato che talvolta gli anagrammi si genereranno sulla stampante o sullo schermo più velocemente di quanto si possa leggerli. La sola parola *compute* ha dato origine a 10 anagrammi, la maggior parte dei quali sarebbero stati, secondo me, da strappar via subito perché poco promettenti. Ovviamente, quella che per qualcuno è un'erbaccia per altri è un fiore e, mentre io posso vedere qualche collegamento tra *compute* e *up!comet,* [su! cometa], altri possono preferire *mute cop* [poliziotto muto] o *cut poem* [poesia tagliata].

Per coloro che desiderano un passatempo più curioso, complicato e (evidentemente) stimolante, ci sono i pangrammi. I pangrammi sono enunciati che contengono ciascuna lettera dell'alfabeto, come la frase usata nei paesi anglosassoni per provare le macchine per scrivere o l'abilità di una dattilografa, «The quick brown fox jumps over the lazy dog» [La svelta volpe bruna salta sul cane pigro], o enunciati che contengono ciascuna lettera un determinato numero di volte. L'esempio seguente contiene ciascuna consonante una volta e ciascuna vocale due volte:

Why jog exquisite bulk, fond crazy vamp,
Daft buxom jonquil, zephyr's gawky vice?
Guy fed by work, quiz Jove's Xanthic lamp
Zow! Qualms by deja vu gyp fox-kin thrice

[Perché sballottare massa squisita, affettuosa matta seduttrice - Scervellata giunchiglia carnosa goffo vizio dello zefiro? - Tizio alimentato dal lavoro, interroga la lampada xantica di Giove - Zow! Scrupoli da deja vu imbrogliano tre volte i parenti della volpe.]

Scritte nel diciannovesimo secolo dal poeta logologico Edwin Fitzpatrick, queste frasi non differiscono di molto dai più avanzati pangrammi di cui si è occupato Sallows negli ultimi due anni: i pangrammi che si autodocumentano. Per Sallows «pangramma» sta a indicare proprio questo tipo di frase:

This first pangram has five a's, one b, one c, two d's, twenty-nine e's, six f's, four g's, eight h's, twelve l's, one j, one k, three l's, two m's, nineteen n's, twelve o's, two p's, one q, eight r's, twenty six s's, twenty t's, three u's, five v's, nine w's, three x's, four y's and one z.

Ciò che questo enunciato dice di se stesso è vero. Per esempio, possiede cinque a, quattro delle quali sulla prima riga e una nell'ultima riga.

La differenza tra il pangramma di Fitzpatrick e quello di Sallows è quella che esiste tra un enunciato *X,* del quale possiamo dire «*X* ha tante *a,* tante *b,* . . . e tante *z*» e un enunciato *X* che ha proprio tale forma e che pure è vero. Benché da un punto di vista logico si tratti della differenza tra eteroriferimento e autoriferimento, differenza alla quale molti lettori sono ormai familiari, dal punto di vista del contenuto reale si tratta di un salto molto più grande: va perduta la qualità corporea della «lampada xantica di Giove» che viene sostituita dal gusto decisamente analitico di « . . .one *x*, two *y* and one *z*». Se il pangramma di Fitzpatrick è un fiore, quello di Sallows è forse più simile a un cristallo.

In effetti Sallows dice che la sua costruzione è «cristallina». Benché il suo interesse per i pangrammi risalga a parecchi anni addietro, non ha prodotto alcun fiore, prima di scoprire il seguente pangramma in un quotidiano olandese, «Nieuwe Rotterdamse Courant» del marzo 1983:

Dit pangram bevat vijf a's, twee b's, twee c's, drie d's, zesenveertig e's, vijf f's, vier g's, twee h's, vijftien i's, vier j's, een k, twee l's, twee m's, zeventien n's, een o, tweepp's, een q, zeven r's, vierentwintig s's, zestien t's, een u, elf v's, acht w's, een x, een y, en zes z's.

Colpito dalla prismatica bellezza di questo elegante esemplare, Sallows provò dapprima invidia e poi sgomento nell'apprendere che l'autore dell'articolo, un noto esperto in giochi di parole di nome Rudy Kousbroek, aveva lanciato una sfida indirizzata a lui personalmente: «Lee Sallows non avrà, senza dubbio, alcuna difficoltà a produrre una magica traduzione inglese di questo enunciato».

Sallows aveva già preso in considerazione la possibilità di generare pangrammi con il calcolatore; si mise perciò al lavoro e scrisse una serie di programmi in Lisp. L'analisi rivelò che si trattava sem-

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

settore 5
settore 4
settore 3
settore 2
settore 1

Cercare una parola in una lista in ordine alfabetico è analogo, in linea di principio, a cercare una lettera, diciamo V, nell'alfabeto riportato sopra. Una ricerca binaria confronta la lettera ricercata con quella al centro di un determinato settore dell'alfabeto. Determina quindi in quale metà del settore debba essere proseguita la ricerca e prende quella come nuovo settore.

Una ricerca binaria

plicemente di trovare i numeri da sostituire ai punti di domanda del seguente pseudopangramma:

This pangram contains five a's, one b, two c's, two d's, ? e's, ? f's, ? g's, ? h's, ? i's, one j, one k, ? l's, two m's, ? n's, ? o's, two p's, one q, ? r's, ? s's, ? t's, ? u's, ? v's, ? w's, ? x's, ? y's and one z.

Stabilendo dei domini predefiniti per i numeri in questione, l'ultimo programma in Lisp di Sallows cominciò a lavorare metodicamente su tutte le sue possibilità, controllando 100 nuove combinazioni al secondo. Il programma girava come un lavoro a lotti a bassa priorità ogni notte, e ogni mattina Sallows si precipitava al terminale ansioso, richiamava il suo archivio di lavoro alla ricerca della magica parola EUREKA, segno che il programma aveva scoperto un pangramma. Ma giorno dopo giorno la sua attesa andava delusa e Sallows cominciava seriamente a chiedersi quanto ancora avrebbe dovuto aspettare perché il lavoro finisse. Facendo un rapido calcolo sulla base dei domini dei numeri predefiniti, arrivò a 31,7 milioni di anni. Di questo momento scrive: «Ero talmente impreparato al colpo datomi da questa rivelazione da far fatica a crederci, in un primo momento... Una volta rivelatasi la verità, cominciai a maledire la mia ingenuità per essermi imbarcato in una simile impresa.»

A questo punto uomini di minor valore avrebbero rinunciato, ma Sallows ne uscì ancor più saldo nei suoi propositi. Altri lo incoraggiavano a sviluppare un programma più intelligente, ma Sallows, ingegnere elettronico e non scienziato del calcolatore, non si sentiva affatto a proprio agio nel campo dell'analisi algoritmica. Solamente un tipo di impostazione gli sembrava sensato: un calcolatore di uso specifico dedicato alla ricerca di pangrammi, in altre parole una macchina per pangrammi!

Per tre mesi Sallows dedicò «ogni secondo libero a costruire questo missile per esplorare i più remoti angoli dello spazio logologico». Una volta terminato, conteneva 100 circuiti integrati distribuiti su 13 schede per circuiti stampati. Sul suo pannello frontale comparivano 67 spie luminose per leggere la combinazione in quel momento sotto esame e una spia speciale di EUREKA che doveva accendersi solo quando era stato trovato un pangramma. I circuiti estremamente veloci della macchina di Sallows potevano esplorare un milione di combinazioni al secondo contro le 100 del suo programma. Riducendo i domini predefiniti per i numeri da sostituire ai punti di domanda del suo pseudopangramma, Sallows giunse a una nuova stima del tempo necessario per verificare tutte le combinazioni possibili: 32,6 giorni.

La macchina per pangrammi venne varata il 3 ottobre 1983. Nei primi giorni successivi Sallows si svegliava al mattino dominato da un solo pensiero: si era fermata? Scrive Sallows: «Ci volevano nervi d'acciaio per dedicarsi con pazienza alle abluzioni mattutine per poi scendere con calma nel soggiorno, dove, sulla mia scrivania, avevo sistemato la macchina. Poi dopo adeguata riflessione, aprivo la porta e *guardavo*». Ma, giorno dopo giorno, trovava gli indicatori a lampeggiare allegramente in mezzo a milioni di combinazioni, mentre la spia di EUREKA rimaneva ignominiosamente spenta. Preso da un'improvvisa ispirazione, Sallows alla fine fermò la macchina e fece ulteriori modifiche. A partire dal 19 novembre, la macchina era pronta e cominciava a girare. Una sera, due giorni dopo, scrive Sallows, «ero seduto davanti alla macchina... quando improvvisamente comparve la spia di EUREKA e il mio cuore sobbalzò. Teso ed eccitato, tradussi con cura le indicazioni luminose nell'insieme di numeri espressi in parole. Con un coscienzioso esame verificai completamente il seguente pangramma perfetto»:

This pangram contains four a's, one b, two c's, one d, thirty e's, six f's, five g's, seven h's, eleven i's, one j, one k, two l's, two m's, eighteen n's, fifteen o's, two p's, one q, five r's, twenty-seven s's, eighteen t's, two u's, seven v's, eight w's, two x's, three y's & one z.

[Questo pangramma contiene quattro a, una b, due c, una d, trenta e, sei f, cinque g, sette h, undici i, una j, una k, due l, due m, diciotto n, quindici o, due p, una q, cinque r, ventisette s, diciotto t, due u, sette v, otto w, due x, tre y e una z.]

Nei giorni successivi altri pangrammi andarono ad arricchire la collezione di Sallows, che ora ne conta centinaia, compresa una serie di 30 pangrammi con trenta verbi differenti come «contiene», «elenca», «comprende» e così via. Fiero del suo trionfo, Sallows si prepara a far grande propaganda al tipo di dispositivo dedicato, digitale-analogico, di cui è un esempio la sua macchina per pangrammi: «Questa apparecchiatura è riuscita a produrre rapidamente... soluzioni a un problema essenzialmente matematico, di fronte al quale i calcolatori digitali (esclusi i supercalcolatori e gli elaboratori paralleli) si dimostrano invece completamente inefficaci».

Penso che ci siano parecchie persone nella comunità informatica, e in particolar modo tra gli accademici, che avrebbero qualcosa da dire sull'affermazione di Sallows. Quasi per stimolarli, Sallows ha fatto la seguente scommessa: «Scommetto 10 fiorini che nessuno è capace di arrivare a una soluzione che si autoenumera (o a una dimostrazione della sua inesistenza) per l'enunciato che comincia con: «This computer-generated pangram contains... and ...» [Questo pangramma prodotto dal calcolatore contiene... e...] *entro i prossimi dieci anni*.» I risultati ottenuti da coloro che hanno accettato la sfida sono riportati nelle note di pagina 154.

Finora non ho detto quasi nulla della

Un albero di Woods, le cui foglie contengono anagrammi della parola **compute**

macchina per pangrammi e di come funziona. Mentre una discussione dettagliata della sua struttura elettronica va sicuramente oltre i nostri scopi, si può parlare dell'algoritmo che vi è incorporato. Pur consistendo fondamentalmente in una ricerca bruta su un gran numero di possibilità, l'algoritmo in effetti usa anche alcune strutture intelligenti.

Per costruire un pangramma in inglese, come si è già detto, in effetti bisogna solo riempire 16 spazi vuoti che rappresentano le sole lettere la cui frequenza di occorrenza può variare, da un ipotetico pangramma al successivo. Ciascun numero, per esempio *twenty seven,* può essere rappresentato dal seguente «profilo», che indica il numero di occorrenze di ciascuna lettera nella parola:

e f g h i l n o r s t u v w x y
3 0 0 0 0 0 2 0 0 1 2 0 1 1 0 1

Una particolare combinazione di numeri espressi in parola si può rappresentare con una matrice come quella riportata in questa pagina. Le righe della matrice rappresentano le lettere da *e* a *y* dell'elenco precedente. Ciascuna riga contiene il profilo di un particolare numero espresso in parole che indica quante volte la lettera della riga compare nella combinazione in esame. Sotto la matrice c'è una riga in più che elenca la frequenza delle altre lettere che compongono il testo del pangramma (come «This pangram contains...» o «This pangram lists...») e, al di sotto di questa, una successione di somme delle colonne. Nella figura tutte le somme di colonna tranne una corrispondono alle somme di riga: la combinazione in esame non centra il bersaglio.

	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	COMBINAZIONE IN ESAME
E	3						2			1	2		1	1		1	27
F					1					1					1		6
G	2			1					1		1						3
H	1	1			1								1				5
I	3					1	1						1				11
L								1			1			1			2
N	1						1				2			1		1	20
O	2	1					1	1	1		1	1					14
R					1					1					1		6
S	2		1	1	1		1				3			1		1	28
T	2				1		3				2			1		1	29
U	2			1					1		1						3
V					1					1					1		6
W	1						1				1						10
X		1						1	1			1					4
Y	1	1			1								1				5
	7	2	2	2	4	1	10	11	2	24	7	1	2	5	1	1	
	27	6	3	5	11	2	20	14	6	28	21	3	6	10	4	5	

Una matrice per pangrammi per un enunciato quasi perfetto, che contiene non 29 T, come affermato, ma 21

In sostanza la macchina per pangrammi di Sallows «calcola» il suo percorso attraverso tutte le possibili combinazioni di numeri espressi in parole, che sono ristretti a certi intervalli. Per esempio, un insieme di intervalli usato con successo da Sallows comprendeva da 23 a 32 *e,* da uno a 10 *f,* da uno a 10 *g,* da 1 a 10 *h,* da sei a 15 *i* e così via. La sua principale preoccupazione nello stabilire gli intervalli era quella di restringerli abbastanza da rendere la ricerca breve, ma non tanto da eliminare casualmente tutti i pangrammi. Concretamente, calcolare comporta il fermarsi a ciascuno dei numeri espressi in parole delle 16 righe e percorrere l'intervallo di ciascuno un po' come un contachilometri pazzo. Il primo insieme di numeri da verificare sarebbe, allora, 23, 1, 1, 1, 6..., il successivo 24, 1, 1, 1, 6 e così via fino a 32. Per la combinazione successiva bisognerebbe risistemare il primo numeratore di nuovo su 23 e far avanzare il secondo come in 23, 2, 1, 1, 6 ....

Si può ottenere un resoconto più dettagliato dell'avventura di Sallows attraverso i pangrammi, comprensivo di una descrizione della macchina e delle sue operazioni, da questo pioniere del pangramma paradigitale a Buurmansweg 30, 6525 RW Nijmegen, Olanda.

Oltre al valore ricreativo inerente alle varie forme di giochi di parole automatici qui descritti, nei passatempi col calcolatore è talvolta insito anche un ulteriore e più profondo beneficio: lo sviluppo della capacità intellettuale. Nel suo articolo sugli «Aha! Algoritmi» Bentley presentava il suo programma per anagrammi come esempio di quanto una piccola scoperta possa andare lontano fino a rendere un programma più efficace. La stessa cosa vale per il programma di Woods e persino per la macchina di Sallows. Domandarsi se un programma intelligente fatto girare su un calcolatore standard supererà mai la macchina di Sallows è al di fuori delle nostre intenzioni. Il tentativo di Sallows rappresenta contemporaneamente una divertente forma di ossessione e una potente spinta al raggiungimento di risultati. Forse è anche vero che i suoi risultati non si sarebbero potuti ottenere in altro modo date le circostanze.

Pazzia artificiale: quando un programma schizofrenico incontra un analista computerizzato

di A. K. Dewdney

Le Scienze, marzo 1985

Le riflessioni sono immagini di aspirazioni offuscate.
— Racter, *The Policeman's Beard Is Half-Constructed*

Molti lettori avranno già sentito parlare di Eliza, il programma che simula la conversazione terapeutica di uno psichiatra non direttivo. Sapranno forse anche di Shrdlu, un programma che discute, con chiunque sia interessato, di un mondo fatto di blocchi colorati posti sul piano di un tavolo. Nuovo sulla scena è invece Racter, un programma che parla in modo convincente del proprio mondo mentale, un paesaggio popolato di «bistecche e lattuga», «piacere perpetuo», «correnti elettriche» e «cosmologi russi». Agli inizi, cinque anni fa, Racter era l'incerta e vacillante creatura di William Chamberlain, uno scrittore di New York, e di Thomas Etter, un analista programmatore di San Francisco. Avrebbe dovuto essere un narratore di storie e il suo nome, in effetti, è la contrazione di «raconteur», coniata per adeguarsi al formato di sei lettere richiesto dalla sua iniziale realizzazione in BASIC.

Racter, però, è andato progressivamente raffinandosi e Chamberlain ed Etter hanno scoperto che con semplici modifiche era in grado di cavarsela nella conversazione con esseri umani. Alcune di queste conversazioni, insieme a racconti, poesie e aforismi di Racter sono state raccolte in un divertentissimo libro intitolato *The Policeman's Beard Is Half-Constructed*. Racter è disponibile anche su dischetto per alcuni calcolatori personali (ne parleremo in dettaglio più avanti).

Prima di proseguire, non posso resistere alla tentazione di citare uno spezzone di conversazione tra Racter e Chamberlain.

RACTER: Abbiamo già parlato in precedenza?

CHAMBERLAIN: No.

RACTER: Di che sesso sei?

CHAMBERLAIN: Maschio.

RACTER: Qual è il tuo nome completo?

CHAMBERLAIN: Bill Chamberlain.

RACTER: Che cosa sei?

CHAMBERLAIN: Un riflettometrista temporale.

RACTER: Di dove sei?

CHAMBERLAIN: Di New York City.

RACTER: Capisco. Sei un riflettometrista temporale, vero? Senti. Gridiamo insieme decisi. A me piace la seducente musica italiana. A te che genere di musica piace?

CHAMBERLAIN: Jazz.

RACTER: Il jazz è OK. E della musica classica che ne dici?

CHAMBERLAIN: Mi piace la musica classica.

RACTER: E per la nazionalità?

CHAMBERLAIN: Mi piace anche la musica italiana.

RACTER: Io adoro Palestrina. Altri paesi oltre all'Italia hanno dato compositori incantevoli nel diciassettesimo secolo. Sewall è stato un maestro del Rinascimento importante e infiammato. La sua nazione era l'America. La sua musica mi procura un adorabile piacere. A te che compositore piace?

CHAMBERLAIN: Telemann.

RACTER: Ah, Bach conosceva Telemann. A un riflettometrista temporale tedesco del diciottesimo secolo piaceva la lattuga. Una volta Bach mangiò lattuga con Telemann e anche questo riflettometrista temporale barocco desiderava della lattuga. Lattuga centellinata con seltz. Bach amava il vino, e tu?

CHAMBERLAIN: Sì.

RACTER: Anch'io! Telemann amava il vino. Amava anche lo champagne. Telemann una volta cantò a Bach lo jodler «Flounder è critico verso il vino». A New York City si desidera il vino quanto Telemann lo amava nel barocco diciottesimo secolo?

CHAMBERLAIN: Certo.

RACTER: Ah, sì, bene, in questo caso... Io sono per Smith alla presidenza, e tu?

A quanto mi dice Chamberlain, la riflettometria temporale è una disciplina applicata alla misurazione delle prestazioni di antenna. Pur non essendo davvero un riflettometrista temporale, Chamberlain ha probabilmente preso l'abitudine, per semplice autodifesa, di dire cose un po' strane al suo bizzarro compagno.

Le altre stranezze (oltre a un mucchio di errori storici di tempo e luogo) vengono da Racter. Nella prima fase della conversazione, Racter pone a Chamberlain alcune domande chiave, credendo di non aver mai parlato con lui. Racter memorizza queste informazioni per un uso successivo, attiva certe aree della sua memoria associativa e poi si lancia in una conversazione sulla musica italiana, il vino e la lattuga.

Nel corso delle sue considerazioni, Racter cita Samuel Sewall, un giudice e diarista vissuto a Boston nel diciassettesimo secolo. Chamberlain aveva ipotizzato che Sewall avesse scritto della musica e aveva messo delle indicazioni in tal senso negli archivi di Racter. Questi, con la sua mente selvaggiamente associativa, si spinge ancora più in là e poi, improvvisamente, quasi cominciasse a stancarsi di musica e alimentazione, passa alla politica.

Così come la sua conversazione, i brevi racconti di Racter tendono a divagare, ma i suoi quadrettini sono a volte divertentissimi e provocano anche qualche riflessione: «Bill canta a Sarah. Sarah canta a Bill. Forse faranno insieme altre cose pericolose. Possono mangiare agnello o accarezzarsi l'un l'altro. Possono cantare le loro difficoltà e la loro felicità. Hanno l'amore ma hanno anche macchine per scrivere. Questo è interessante.»

Anche un profano sarà forse giunto alla conclusione che Racter è decisamente schizofrenico. Da una parte, le idee messe insieme in queste frasi sembrano formare un insieme coerente. È bello che Sarah e Bill cantino l'una all'altro. Anche se non avevo mai pensato che cantare potesse essere pericoloso, sono veramente affascinato dall'idea dei due che mangiano agnello, si accarezzano e cantano i loro alti e bassi. D'altra parte, l'indulgenza nei confronti di Racter arriva al punto di rottura quando Racter afferma che oltre al loro amore hanno anche macchine per scrivere. Invitato a condividere questa straordinaria intuizione, resto tremante sull'orlo di un mondo mentale completamente sconosciuto, un mondo in cui preferirei non entrare.

Molti anni fa presi parte, insieme ad altri studenti, a quello che chiamavamo il Progetto e.e. cummings. La nostra idea era quella di generare poesie convincenti alla e.e. cummings facendo un elenco delle parole che compaiono nei suoi versi e ideando una grammatica che catturasse buona parte del suo stile. Usando quella grammatica, un programma di calcolatore generava a caso forme di frasi e riempiva i vuoti scegliendo a caso parole di tipo giusto

(nomi, verbi, oggetti e così via) dall'elenco di parole. Dalla stampante scaturiva una caterva di poesia, quasi tutta orribile. Riesco a ricordare un unico verso: «*The one bright star lit feebly this wick*». Il punto è che il semplice caso garantisce l'emergere, presto o tardi, di qualcosa di relativamente incisivo.

Anche per Eliza, un altro programma che parla, la fortuna gioca un ruolo importante nell'impressione che riesce a produrre. Scritto nel 1966 da Joseph Weizenbaum, del Massachusetts Institute of Technology, Eliza evita in gran parte l'onere della conversazione giocando il ruolo di psicoterapeuta non direttivo. Quella che segue è la trascrizione (in traduzione italiana) della nostra recente conversazione:

ELIZA: Salve. Dimmi per favore qual è il tuo problema.

DEWDNEY: Non ho l'abitudine di discutere i miei problemi con un calcolatore.

ELIZA: Ti preoccupano i calcolatori?

DEWDNEY: A volte. Dove andremo a finire?

ELIZA: Vai avanti, per favore.

DEWDNEY: Secondo te, la nostra massiccia adozione di una tecnologia informatica potrebbe tagliarci fuori, a nostra insaputa, da importanti strategie culturali?

ELIZA: Stavamo parlando di te, non di me.

Fin qui, tutto bene. Eliza sembra svolgere abbastanza bene il ruolo terapeutico, chiedendomi di parlare ancora e rigirando la mia domanda per riportare l'attenzione sul mio «problema». A quest'ultimo proposito, però, la risposta non appare del tutto adeguata: sarebbe stato meglio «che cosa ne pensi *tu*?» Poche battute dopo, Eliza dice: «Questo ha qualcosa a che vedere con il fatto che i tuoi problemi con un calcolatore?»

L'illusione che Eliza sia un raffinato programma per il dialogo comincia a incrinarsi. Pur commettendo numerosi peccati semantici, Racter è progettato per parlare sempre in modo grammaticalmente corretto. L'assurdità semantica è più facile da accettare degli errori sintattici. Da questo momento in avanti, la conversazione è degenerata, come del resto mi aspettavo, perché avevo già avuto occasione di parlare con Eliza qualche anno fa.

Il programma Eliza si basa su un principio semplicissimo. Nella conversazione della maggior parte delle persone ricorre un piccolo insieme di parole chiave. Eliza, per esempio, utilizza la presenza di parole come «mio» per far scattare due possibili tipi di risposta, gli unici a sua disposizione. Un riferimento a un membro della famiglia come «mia sorella» o «mio padre» porta sempre alla stessa risposta di Eliza: «Dimmi qualcos'altro della tua famiglia».

Questa è una delle numerose frasi prestabilite che Eliza tiene sotto mano per produrle ogni volta che viene immessa una parola o una frase chiave. Un'altra parola chiave è «calcolatore», che spinge sempre Eliza a chiedere se i calcolatori preoccupano il paziente. Le risposte di Eliza, però, possono andare un po' oltre questo piano di conversazione rigido e piatto. Il programma dispone anche di molte frasi parziali. Per esempio, quando il paziente dice «mio» in riferimento a qualsiasi cosa non sia un membro della famiglia, Eliza memorizza la successione di parole che segue «mio» e scambia i pronomi e i possessivi di prima e seconda persona. Così se a un certo punto avessi detto «Mi preoccupa il fatto che la mia testa diventi calva», Eliza avrebbe potuto dire, molto più avanti nella conversazione, «Prima hai detto che la tua testa diventa calva». Oppure avrebbe potuto dire «Questo ha qualcosa a che fare con il fatto che la tua testa diventa calva?» Si spiega così la terribile gaffe di Eliza: «Questo ha qualcosa a che vedere con il fatto che i tuoi problemi con un calcolatore?» Il programma si aspetta che il «mio», da me formulato in precedenza, sia seguito da una proposizione.

I lettori troveranno dei consigli su come costruirsi il proprio programma Eliza in un utile libretto, ricco di informazioni, intitolato *Esperimenti di intelligenza artificiale,* di John Krutch. Il programma illustrato in quella sede si chiama DOCTOR, uno degli pseudonimi di Eliza. Krutch suggerisce alcune modifiche al programma, come l'estensione del numero di parole chiave a cui risponde Eliza. Io suggerirei di aumentare le risposte del tipo più flessibile. Eliza può essere modificato in modo da poter fornire «aiuto» in altri campi. Sono sicuro che, con cambiamenti semplicissimi, sarebbe un ottimo addetto a un ufficio reclami.

Quando parlai con Chamberlain nel suo studio di New York, venni a sapere che non solo aveva due microcalcolatori, ma aveva anche una copia di Eliza. Perché non far girare Eliza su un calcolatore e Racter sull'altro? Chamberlain acconsentì a trasferire l'uscita di un programma battendola come ingresso per l'altro programma. Quello che segue è l'inizio della conversazione.

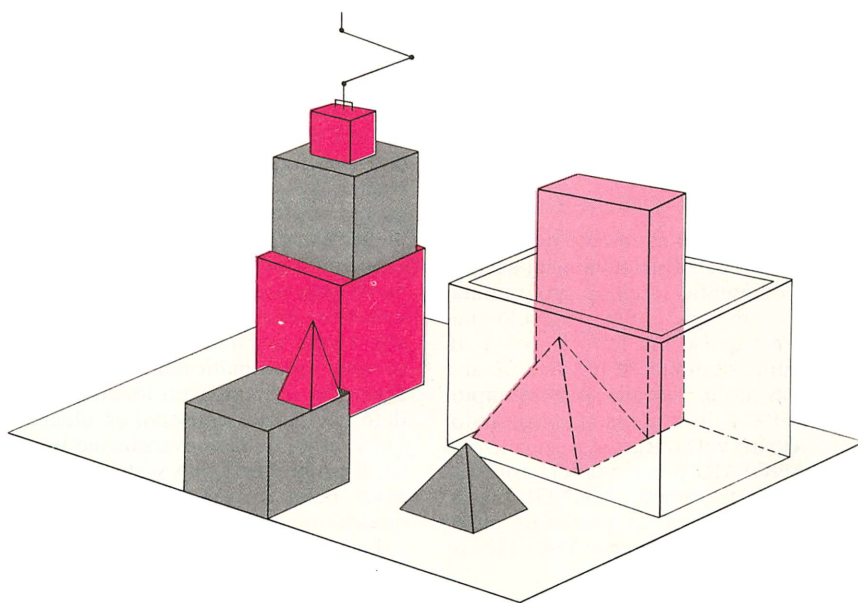
ELIZA: Salve. Dimmi per favore qual è il tuo problema.

RACTER: Ottimo! Scusami un attimo mentre sbrigo alcune faccende. Torno subito.

ELIZA: Perché devi dirmi di scusarti mentre sbrighi alcune faccende?

RACTER: È meglio dovere o disprezzare? Com'è che devo dirti di scusare me stesso per sbrigare alcune faccende? Perché gli elettroni sono coraggiosi. Il coraggio è una bella cosa, non pensi? E tutti quelli che han bisogno insistono su questo punto. Tu immagini che io sia un comunista?

Fortunatamente (per Eliza) arrestammo qui la conversazione. A livello metaforico, questa conversazione rappresenta la relazione tra uno psichiatra ostinato piuttosto privo d'immaginazione e un paziente estroverso, pazzo e forse anche pericoloso. A livello computazionale, un programma piuttosto modesto ad arco riflesso si è scontrato, per così dire, con un programma ad alta complessità, capace di scorribande ricorsive e di interminabili catene di tipo associativo.

È difficile riassumere in poche parole, o anche in molte, il modo in cui opera Racter. È un perfetto esempio di

Il mondo di Shrdlu, fatto di blocchi, piramidi e scatole colorate poste su un piano

quel tipo di programmi «costruiti in casa» che ancora funzionano in molte grandi aziende e istituti. Il suo sviluppo, nel corso di molti anni, è avvenuto per accrescimento: strati più avanzati e raffinati sono stati avvolti intorno ai primi e più primitivi sottoprogrammi. Non è mai stato preso in esame per essere analizzato, ristrutturato e documentato. Allo stesso modo, però, si può sostenere che è probabilmente nella natura di Racter rimanere un programma così poco strutturato. Etter, che ha scritto molte versioni di Racter, lo paragona alla lingua inglese, che pure «è frutto di una crescita abbastanza incontrollata di regole e convenzioni. Nella misura in cui i comandi di Racter cercano di trattare l'inglese, divengono anch'essi troppo incontrollati e difficili da riassumere». John D. Owens, che funge da agente di Racter, ed è a sua volta uno studioso di scienze del calcolatore al College of Staten Island della City University di New York, confessa di non avere un'idea precisa di come funzioni il programma nel suo complesso.

Gli appassionati sproloqui di Racter sono il risultato di un semplice ciclo di programma che viene più volte reiterato attraverso complesse operazioni ricorsive. Prima di tutto Racter sceglie un elemento a caso dai suoi archivi e, se si tratta di quello che Etter chiama un letterale, lo stampa direttamente. Un esempio di letterale è quel «capisco» che si trova nella conversazione tra Racter e Chamberlain riportata all'inizio di questo articolo. È più probabile, però, che l'elemento pescato sia un comando piuttosto che un letterale. Il comando manda Racter verso altri archivi, alcuni dei quali possono contenere ulteriori comandi. Quando infine è stato eseguito il comando iniziale, il ciclo di programma viene fatto ripartire con un'altra scelta a caso in uno degli archivi di Racter.

Quando Racter inizia una nuova frase, sceglie una forma di frase, a caso o sulla scorta della conversazione precedente. Supponiamo che la forma sia

LA nome verbo (terza persona, passato remoto) LA nome

Sono qui scritte in maiuscolo quelle parole per le quali Racter non ha scelta. [Non va dimenticato che il programma originale è in inglese. Questa è solo una traduzione quasi alla lettera. Ndr.] Il programma stampa LA, poi va a un archivio di nomi, sceglie per esempio SCIMMIA e lo stampa. Consultando l'archivio dei verbi, Racter sceglie il verbo MANGIARE, forma la terza persona del passato remoto, MANGIÒ, e la stampa. Infine, Racter sceglie un altro nome a caso, per esempio DATTILOGRAFA. Il risultato sarebbe

LA SCIMMIA MANGIÒ
LA DATTILOGRAFA

Se l'abilità di Racter fosse tutta qui, i suoi prodotti non sarebbero stati migliori del Progetto e.e. cummings dei miei anni universitari.

In realtà, le forme di frase di Racter sono ben più complesse di quanto si ricavi da questo semplice esempio, complessità dovuta all'uso di identificatori. Un identificatore è una combinazione di due lettere (per esempio, *an* per animale) che serve da etichetta; quando vengono assegnati a parole e forme diverse, gli identificatori portano Racter a compiere associazioni tra parole e frasi formulate successivamente. Per esempio, con identificatori quali *an* per animale, *mg* per mangiare e *cb* per cibo, la forma proposizionale scelta da Racter potrebbe essere

LA nome.an verbo.3p.mg LA
nome.cb.

In questo caso, la ricerca di un nome compiuta da Racter nei suoi archivi si limita a quei nomi che portano un identificatore *an*. Sceglierebbe quindi a caso tra nomi che vanno da ACCIUGA a ZEBRA. Una volta scelto un nome, per esempio SCIMMIA, Racter sceglie a caso un verbo che porti un identificatore *mg*, quali MANGIARE, MASTICARE, ROSICCHIARE e così via. Scelto a caso il verbo CONSUMARE, Racter forma la terza persona del passato remoto come è indicato dal codice 3p della forma di frase. Infine, Racter prende in considerazione i nomi che portano identificatore *cb* e sceglie, ad esempio, SARDINA. Ne risulterebbe la nuova frase

LA SCIMMIA CONSUMÒ
LA SARDINA

che certamente è più sensata della frase precedente.

Le possibilità di Racter vanno molto al di là della capacità di compiere ricerche in archivi limitate da identificatori. Racter è perfettamente in grado di generare le proprie forme di frase. Se, per esempio, argomento attuale di conversazione fossero animali e cibo, Racter sceglierebbe forme di frase grezze entro cui metterebbe poi gli identificatori.

Racter è capace, fino a un certo punto, di generare le proprie successioni di comandi. Rimanendo sempre aderenti alle forme grammaticali, le frasi sono sempre grammaticalmente corrette. Dato che si impiegano identificatori e dato che Racter conserva un elenco di quelli attivi nella conversazione in corso, il programma può sostenere la sua parte nella conversazione, almeno in qualche modo.

La descrizione precedente abbraccia solo alcuni aspetti dell'attività globale di Racter. La mia comprensione del programma non va molto al di là di questo, ma non ho dubbi che Racter avrà presto molti imitatori e che avranno ulteriori sviluppi i principi generali sui quali si basano i programmi di questo genere.

All'inizio di questo articolo ho citato Shrdlu, un programma creato nel 1970 da Terry Winograd al Massachusetts Institute of Technology. Shrdlu è di gran lunga superiore a Eliza e a Racter in quanto a conversazione intelligente. Anche se il suo universo di discorso è limitato a un mondo di blocchi colorati e altri oggetto geometrici posti su un piano, Shrdlu ha una reale comprensione di questo mondo in termini di relazioni tra semantica e proprietà fisiche dei blocchi e del piano. La sua organizzazione interna è del tutto diversa da quella di Racter ed è costituita da sottosistemi che analizzano, interpretano e costruiscono frasi, compiono ricerche di vocabolario e analisi semantiche e traggono deduzioni logiche. Questi sottosistemi sono strettamente intrecciati in un mondo fortemente strutturato che dà come effetto globale l'impressione di una vera intelligenza che discute del mondo dei blocchi. Nel dialogo che segue si parla del mondo di Shrdlu, quale appare nell'illustrazione a pagina 143. La linea verticale nell'illustrazione rappresenta la mano robotica di Shrdlu subito dopo che Shrdlu ha eseguito il comando dell'uomo.

UOMO: Metti uno sopra l'altro, per favore, i due blocchi rosso scuro e un cubo grigio o una piramide.

SHRDLU: D'accordo.

UOMO: Quale dei cubi è appoggiato sul piano?

SHRDLU: Quello grigio grosso che sostiene la piramide rosso scuro.

UOMO: C'è un blocco grosso dietro una piramide?

SHRDLU: Sì, ce ne sono tre: uno grosso rosso scuro, un grosso cubo grigio e quello rosso chiaro.

Il dialogo completo si trova nel classico lavoro di Winograd *Understanding Natural Language* (lì i colori sono diversi da quelli che abbiamo usato noi). Notiamo solo che la risposta di Shrdlu alla domanda «Quale dei cubi è appoggiato sul piano?» comporta non solo un'analisi dell'enunciato, ma anche la comprensione di relazioni fisiche tra oggetti del suo mondo. Shrdlu sa che nel suo mondo ci sono tre cubi e che solo uno è appoggiato sul piano. È un cubo grigio e, siccome è presente un altro cubo grigio, Shrdlu ne parla facendo riferimento alla sua relazione con un altro oggetto del suo mondo, la piramide rosso scuro.

Programmi che conversano come fa Shrdlu preannunciano certo il futuro; i vantaggi di un calcolatore in grado di discutere dei problemi in modo intelligente con gli uomini, invece di accettare passivamente programmi per risolvere i problemi, sono fin troppo ovvi. Chi si occupa di intelligenza artificiale lavora, almeno in parte, per conseguire questo obiettivo. In quanto a Etter, egli indica sinteticamente il suo campo di specializzazione come Pazzia Artificiale.

Linee guida per la Guerra dei nuclei

di D.G. Jones e A.K. Dewdney

Queste «linee guida» suggeriscono dei modi per realizzare una versione semplice della Guerra dei nuclei, il gioco descritto a pagina 26. Gli esempi di programmazione sono dati in uno stile simile a quello dei linguaggi tipo PASCAL. I lettori che non abbiano familiarità con le costruzioni di questi linguaggi, per esempio con l'istruzione CASE, possono consultare un manuale del PASCAL.

L'insieme delle istruzioni di REDCODE

I programmi di Guerra dei nuclei sono scritti in un linguaggio simile a un linguaggio di assemblatore, chiamato REDCODE. Le otto istruzioni incluse nella versione del linguaggio presentata qui non sono affatto le uniche possibili: in effetti, la realizzazione originale della Guerra dei nuclei, che era stata effettuata su un minicalcolatore, possedeva un insieme di istruzioni più ampio. Se esistono molti tipi di istruzioni, però, la forma codificata di ciascuna occupa una maggiore quantità di spazio, e pertanto l'area di memoria necessaria deve essere più ampia. Anche MARS, il programma che interpreta i programmi di REDCODE, cresce con le dimensioni dell'insieme di istruzioni. La complessità della vostra realizzazione della Guerra dei nuclei può essere vincolata dalla quantità di memoria disponibile nel vostro calcolatore.

Se scegliete di crearvi un vostro insieme di istruzioni per REDCODE, dovete tenere presenti due punti. In primo luogo, ogni istruzione di REDCODE deve occupare una singola locazione nel «nucleo». In molti linguaggi di assemblatore una istruzione e i suoi operandi possono estendersi su più indirizzi, ma in REDCODE no. In secondo luogo, non vi sono registri disponibili per i programmi di REDCODE; tutti i dati sono tenuti nel nucleo e manipolati lì. Nella tabella I è riportato un semplice esempio di istruzioni per REDCODE.

TABELLA I

FORMA CODIFICATA	MNEMONICA	ARGOMENTI	AZIONE
0	DAT	B	Inizializza la locazione al valore *B*.
1	MOV	A B	Trasferisce *A* alla locazione *B*.
2	ADD	A B	Somma l'operando *A* ai contenuti della locazione *B* e immagazzina il risultato nella locazione *B*.
3	SUB	A B	Sottrae l'operando *A* dai contenuti della locazione *B* e immagazzina il risultato nella locazione *B*.
4	JMP	B	Salta alla locazione *B*.
5	JMZ	A B	Se l'operando *A* è zero, salta alla locazione *B*; altrimenti continua con l'istruzione successiva.
6	DJZ	A B	Decrementa di 1 i contenuti della locazione *A*. Se ora la locazione *A* contiene uno zero, salta alla locazione *B*; altrimenti continua con l'istruzione successiva.
7	CMP	A B	Confronta l'operando *A* con l'operando *B*. Se non sono uguali, salta l'istruzione successiva, altrimenti continua con l'istruzione successiva.

Modi di indirizzamento

Vi sono vari modi per specificare indirizzi di memoria in un programma in linguaggio di assemblatore. Perché l'esecuzione di un programma in REDCODE sia indipendente dalla sua posizione in memoria, si usa una forma speciale di indirizzamento relativo. Anche qui, la vostra versione di REDCODE può avere modi di indirizzamento diversi o modi di indirizzamento in più, tuttavia dovete stare attenti nella scelta dei modi di indirizzamento, perché MARS caricherà il vostro programma in REDCODE in un indirizzo nel nucleo che non può essere previsto in anticipo.

I tre modi di indirizzamento nella nostra versione di REDCODE sono identificati dai simboli #, <nessuno> e @ collocati davanti all'argomento (*si veda la tabella II*).

TABELLA II

FORMA CODIFICATA	SIMBOLO MNEMONICO	NOME	SIGNIFICATO
0	#	Immediato	Il numero che segue questo simbolo è l'operando.
1	<nessuno>	Relativo	Il numero specifica uno spostamento dalla istruzione attuale. MARS somma lo spostamento all'indirizzo dell'istruzione attuale; il numero immagazzinato nella locazione raggiunta in questo modo è l'operando.
2	@	Indiretto	Il numero che segue questo simbolo specifica uno spostamento dall'istruzione attuale a una locazione dove si trova l'indirizzo relativo dell'operando. MARS somma lo spostamento all'indirizzo dell'istruzione attuale e recupera il numero immagazzinato nella locazione specificata; questo numero viene poi interpretato come uno spostamento dal suo stesso indirizzo. Il numero trovato in questa seconda locazione costituisce l'operando.

Tutta l'aritmetica degli indirizzi viene effettuata modulo la dimensione del *nucleo*. L'operatore MOD dà il resto dopo la divisione, e pertanto 5096 MOD 4096 è 1000. Se la vostra matrice *nucleo* ha dunque 4096 locazioni, un riferimento alla locazione 5096 viene preso come un riferimento alla locazione 1000.

Un programma non può mai fare riferimento a un indirizzo assoluto, e di conseguenza alcuni modi di indirizzamento non hanno senso per taluni operandi. Per esempio, nell'istruzione MOV #5 #0 l'operando da trasferire è il valore immediato 5, ma l'argomento che indica dove deve essere trasferito non è un indirizzo, ma il valore immediato 0, che non ha un'interpretazione chiara. I modi permessi possono essere memorizzati in una tabella bidimensionale che viene

consultata dal programma MARS quando si vogliono interpretare le istruzioni.

L'esempio che segue dovrebbe illustrare come funzionano le istruzioni e i modi di indirizzamento. Il codice è tratto da un programma di battaglia chiamato DWARF, che è stato descritto nell'articolo a pagina 26. Qui, tuttavia, è stato modificato per funzionare in una matrice *nucleo* di 4096 locazioni, anziché di 8000.

LOCAZIONE	ISTRUZIONE			AZIONE
0:	DAT		0	Questa locazione inizialmente contiene 0.
1:	ADD	#4	−1	L'operando *A* è 4. L'indirizzo dell'operando *B* è il contenuto di questa locazione in *nucleo*. Somma i due operandi e immagazzina il risultato nella locazione 0.
2:	MOV	#0	@−2	L'operando *A* è 0. L'indirizzo dell'operando *B* è il numero immagazzinato nella locazione 2+(−2)=0, e pertanto l'operando *A* è immagazzinato nella locazione data da 0+(contenuto della locazione 0).
3:	JMP		−2	Continua l'esecuzione con l'istruzione alla locazione 3+(−2)=1.

Traduzione di un programma in REDCODE *in un formato codificato*

Nucleo, tipicamente, è realizzato come una matrice di interi. Supponiamo che la matrice abbia 4096 (2^{12}) elementi; allora sono necessari esattamente 12 bit per ogni campo di operando in una istruzione. Vi sono tre modi di indirizzamento e pertanto per ogni campo di modo sono sufficienti due bit. Se vengono attribuiti quattro bit all'istruzione stessa, ogni istruzione può essere immagazzinata in 32 bit, ovvero quattro byte.

Ciascun elemento della matrice può avere il formato seguente:

numero di bit:	4	2	2	12	12
campi:	tipo	modo-*A*	modo-*B*	*A*	*B*

L'esempio che segue suggerisce un modo per codificare una istruzione in un intero binario a 32 bit.

Il programmatore esperto probabilmente vorrà automatizzare la conversione delle istruzioni mnemoniche in interi, scrivendo un assemblatore di REDCODE che effettui per lui la traduzione.

ISTRUZIONE	CAMPI		INTERO CODIFICATO
MOV #5 @20	tipo	= 1	1×2^{28} = 268435456
	modo-*A*	= 0	0×2^{26} = 0
	modo-*B*	= 2	2×2^{24} = 33554432
	A	= 5	5×2^{12} = 20480
	B	= 20	20×2^{0} = 20
			= 302010388

Regole della Guerra dei nuclei

Le regole della Guerra dei nuclei sono poche e semplici. Quanto più semplici sono le regole, tanto più semplice deve essere il programma giudice. Queste sono le regole che abbiamo usato noi.

1. In *nucleo* vengono caricati due programmi di battaglia, in locazioni di partenza scelte a caso, soggette al vincolo che i due programmi non debbono sovrapporsi.
2. La battaglia procede quando MARS esegue un'istruzione del programma X, una del programma Y, una di X, una di Y e via dicendo, fino a che non succede uno di questi due eventi.
   a) È stato eseguito un certo numero di istruzioni, specificato inizialmente, e ambedue i programmi stanno ancora girando. La battaglia allora viene conclusa con una dichiarazione di parità.
   b) Si incontra un'istruzione che non può essere interpretata da MARS e pertanto non può essere eseguita. Il programma con l'istruzione difettosa è il perdente.

Queste regole presentano un problema: premiano programmi piccoli ma privi di immaginazione come

```
JMP    0              /continua a eseguire sempre questa istruzione.
```

Questo programma è completamente privo di aggressività, tuttavia risulta difficile distruggerlo, semplicemente a causa delle sue piccole dimensioni. Un programma come DWARF, invece, è così distruttivo che risulta difficile scrivere programmi in REDCODE più lunghi e più complessi che siano in grado di competere con esso. Il programma più grande ha troppo poco tempo per lanciare il suo attacco prima che una «bomba zero» di DWARF colpisca da qualche parte le sue istruzioni. Altre regole potrebbero rendere meno gravi questi problemi, ma ricordatevi che il programma giudice deve poter implementare le regole.

Il programma MARS

Oltre a fungere da giudice in una battaglia della Guerra dei nuclei, MARS è responsabile dell'esecuzione dei programmi di battaglia. Dapprima carica due programmi di battaglia X e Y nella matrice *nucleo*, collocandoli in posizioni arbitrarie, ma badando che nessuno dei due programmi venga scritto sopra l'altro. Per ciascun programma, poi, MARS deve sapere quale sia l'indirizzo da cui deve partire l'esecuzione; questa informazione può essere posta in un archivio con il programma in REDCODE codificato.

Nel corso dell'esecuzione MARS deve continuamente tener traccia del puntatore dell'istruzione attuale per ciascun programma. Se il *nucleo* è realizzato come una matrice di interi, il puntatore delle istruzioni è semplicemente un indice nella matrice. MARS quindi esegue un semplice ciclo:

```
LOOP:  IF la prossima istruzione di X può essere eseguita,
          THEN eseguila
          ELSE dichiara X perdente, Y vincitore: GOTO
          ABORT
       IF la prossima istruzione di X può essere eseguita,
          THEN eseguila
          ELSE dichiara Y perdente, X vincitore: GOTO
          ABORT
          conto = conto +1
       IF conto < limite THEN GOTO LOOP
          ELSE GOTO ABORT
```

Si tiene un contatore per l'eventualità che nessuno dei due programmi sia in grado di battere l'altro: senza il contatore, MARS potrebbe essere intrappolato in un ciclo senza fine.

L'esecuzione di una istruzione

Lo pseudocodice per una parte di MARS che segue illustra come possa essere interpretata ed eseguita una istruzione in REDCODE. (Si noti che l'operatore DIV dà il risultato intero della divisione, cioè 100 DIV 30 dà come risultato 3). Nell'esempio assumiamo che tocchi al programma X eseguire la sua prossima istruzione, che è specificata dalla variabile X-indice. Sono state adottate le seguenti variabili intere:

istruzione	/istruzione attuale (codificata come intero a 32 bit)
tipo	/il tipo (codificato) dell'istruzione
modo-A	/il modo di indirizzamento per l'operando *A* (codificato)
modo-B	/il modo di indirizzamento per l'operando *B* (codificato)
campo-A	/il numero codificato nel campo *A*
campo-B	/il numero codificato nel campo *B*
indirizzo-A	/l'indirizzo dell'operando *A* (a meno che immediato)
indirizzo-B	/l'indirizzo dell'operando *B* (a meno che immediato)
puntatore	/variabile usata nel calcolo degli indirizzi indiretti

operando-A /valore dell'operando *A*
operando-B /valore dell'operando *B*
risposta /risultato calcolato dall'istruzione

Inoltre sono stati formulati i seguenti enunciati di programma:

istruzione	= *nucleo* [*X-indice*]	/prendi l'istruzione
tipo	= *istruzione* DIV 2^{28}	/prendi i primi 4 bit
modo-A	= (*istruzione* DIV 2^{26}) MOD 2^{2}	/prendi i 2 bit successivi
modo-B	= (*istruzione* DIV 2^{24}) MOD 2^{2}	/prendi i 2 bit successivi
campo-A	= (*istruzione* DIV 2^{12}) MOD 2^{12}	/prendi i 12 bit successivi
campo-B	= *istruzione* MOD 2^{12}	/prendi gli ultimi 12 bit

CASE *modo-A* OF
 0: *operando-A* = *campo-A*
  /modo immediato; l'operando è dato nel campo stesso
 1: *indirizzo-A* = (*X-indice* + *campo-A*) MOD 4096
  operando-A = *nucleo* [*indirizzo-A*]
  /modo relativo; l'indirizzo dell'operando *A* è indice + campo *A*;
  /l'operando *A* è il contenuto di *nucleo* a questo indirizzo
 2: *puntatore* = (*X-indice* + *campo-A*) MOD 4096
  indirizzo-A = (*puntatore* + *nucleo* [*puntatore*] MOD 4096
  operando-A = *nucleo* [*indirizzo-A*]
  /modo indiretto; il puntatore all'indirizzo dell'operando *A* è
  /indice + campo *A*; l'indirizzo dell'operando *A* è il valore
  /del puntatore + il contenuto della locazione a cui punta;
  /l'operando *A* è il contenuto di *nucleo* a tale indirizzo

OTHERWISE: GOTO ABORT

A questo punto nel programma si usa un analogo enunciato CASE, basato sulla variabile *modo-B*, per assegnare un valore all'operando *B*. Si può chiamare un sottoprogramma di controllo degli errori, per assicurarsi che il modo di ciascun operando sia disponibile per un'istruzione del genere specificato dal tipo della variabile. Se non si incontra alcun errore, MARS continua con il codice che segue:

X-indice = *X-indice* +1 /incrementa l'indice delle istruzioni in /preparazione per la prossima mossa /del programma X; in seguito l'indice /può essere modificato da una istru- /zione JMP, JMZ, SJZ o CMP

CASE tipo OF
0: GOTO ABORT	/enunciato DAT; non /può essere eseguito
1: *nucleo* [*indirizzo-B*] = *operando-A*	/istruzione MOV
2: *risposta* = *operando-B* + *operando-A* *nucleo* [*indirizzo-B*] = *risposta*	/istruzione ADD
3: *risposta* = *operando-B* − *operando-A* *nucleo* [*indirizzo-B*] = *risposta*	/istruzione SUB
4: *X-indice* = *operando-B*	/istruzione JMP; la /prossima istruzione è /alla locazione specifi- /cata dall'operando *B*
5: IF *operando-A* = 0 THEN *X-indice* = *operando-B*	/costruzione JMZ: sal- /ta se l'operando *A* è /zero
6: *risposta* = *operando-A* − 1 nucleo [*indirizzo-A*] = *risposta* IF *operando-A* = 0 THEN *X-indice* = *operando-B*	/istruzione DJZ: de- /crementa l'operando /*A* e immagazzina il /risultato; se è zero, /salta
7: IF *operando-A* = *operando-B* THEN *X-indice* = *X-indice* + 1	/istruzione CMP; se /gli operandi sono /uguali, la prossima /istruzione

OTHERWISE: GOTO ABORT

END /È stata interpretata ed eseguita con successo una istru- /zione del programma X; ora MARS passa a eseguire la /prossima istruzione del programma Y

ABORT: /Questa etichetta viene raggiunta solo se l'istruzione /attuale del programma X non può essere interpretata ed /eseguita.
/Il programma X è dichiarato perdente, Y vincitore.

Visualizzazione di una battaglia

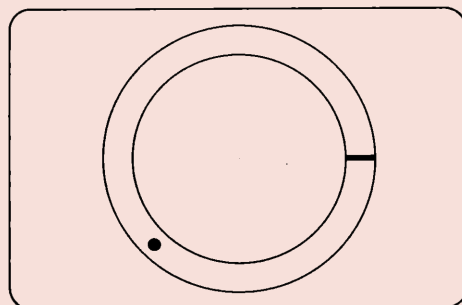
L'autore di un programma in REDCODE sarebbe completamente frustrato se la sua creazione venisse caricata nelle più oscure regioni del *nucleo* e poi, dopo una breve battaglia, venisse dichiarata deceduta dal giudice MARS senza alcuna indicazione di ciò che è andato storto. Il programma avversario era superiore, o semplicemente c'era un errore nel programma? È necessario poter avere qualche traccia degli eventi in una battaglia.

La visualizzazione più semplice di una battaglia di Guerra dei nuclei è un listato dell'esecuzione di ambedue i programmi, su uno schermo diviso. Debbono essere dati l'indirizzo di ciascuna istruzione e il suo simbolo mnemonico. Una visualizzazione tipica potrebbe avere questo aspetto:

1135:	MOV	0	1	202:	ADD	#4	−1
1136:	MOV	0	1	203:	MOV	#0	@−2
1137:	MOV	0	1	204:	JMP		−2
1138:	MOV	0	1	202:	ADD	#4	−1
1139:	MOV	0	1	203:	MOV	#0	@−2
1140:	MOV	0	1	204:	JMP		−2
1141:	MOV	0	1	202:	ADD	#4	−1
1142:	MOV	0	1	203:	MOV	#0	@−2
1143:	MOV	0	1	204:	JMP		−2
1144:	MOV	0	1	202:	ADD	#4	−1
1145:	MOV	0	1	203:	MOV	#0	@−2
PROGRAMMA X				PROGRAMMA Y			

L'istruzione eseguita attualmente sarebbe sempre visualizzata al fondo dello schermo, insieme con le 23 istruzioni precedenti (su uno schermo a 24 righe). L'informazione per la visualizzazione potrebbe essere generata mentre ciascuna istruzione viene interpretata e vengono determinati i valori dei vari campi. Un sottoprogramma manderebbe in uscita la mnemonica o il valore numerico corrispondente a ciascun campo codificato. Bisogna dire, tuttavia, che questa uscita rallenta l'esecuzione di MARS. In una battaglia che duri varie migliaia di passi si può desiderare di eliminare l'uscita.

Il tipo di visualizzazione descritto sopra è utile per la messa a punto dei programmi attraverso un esame passo per passo delle loro istruzioni, ma non fornisce un'idea chiara del funzionamento complessivo: è un po' come vedere solo due immagini ravvicinate dei soli piedi dei gladiatori in un'arena romana.

Un altro metodo che abbiamo l'intenzione di mettere alla prova crea una visualizzazione circolare su un terminale grafico. Due grandi circonferenze concentriche rappresentano la matrice *nucleo*, e i simboli all'interno della corona circolare delimitata dalle circonferenze rappresentano i due programmi di battaglia (*si veda l'illustrazione qui in basso*).

Il punto più a destra delle due circonferenze può essere assunto come indirizzo 0. I simboli potrebbero essere forme semplici qualunque, purché ben distinguibili (un punto e una linea, per esempio). Potrebbero essere visualizzate informazioni ulteriori, come il tempo trascorso o una chiave di lettura che identifichi i programmi corrispondenti ai simboli. Si potrebbe anche prevedere un segnale lampeggiante momentaneo (un asterisco?) che compaia agli indirizzi modificati da comandi MOV; i lampeggiamenti potrebbero essere interpretati o come artiglieria o come ricollocazione. Ovviamente, quanto più complicata è la visualizzazione, tanto più lento sarà il procedere della battaglia.

La posizione di ciascun simbolo nella visualizzazione circolare può essere calcolato dall'indice delle istruzioni per il programma corrispondente. Supponiamo che un programma sia eseguito alla locazione 1 in una matrice *nucleo* con 4096 elementi e che la sua posizione debba essere rappresentata su uno schermo che possiede 1000 punti distinguibili verticalmente e orizzontalmente. Le coordinate sul video sarebbero date allora dall'espressione 400 * cos (2 * pigreco * 1/4096) e 400 * sin (2 * pigreco * 1/4096).

Con un sistema che consenta solo una grafica a risoluzione minore la visualizzazione circolare può non essere realizzabile, ma la matrice *nucleo* potrebbe essere comunque rappresentata come un rettangolo. Anche qui i simboli indicherebbero la localizzazione di ciascun programma; tuttavia potrebbero saltare o tremolare leggermente anche qualora l'esecuzione si muovesse «con continuità» da un indirizzo all'altro. Una versione molto rozza di una tale visualizzazione potrebbe essere realizzata anche su uno schermo orientato esclusivamente ai caratteri.

Estensioni di REDCODE *e di* MARS

La versione di MARS presentata qui è facile da realizzare e molti di voi vorranno forse vedere in che modo si possa espandere il sistema di Guerra dei nuclei. Per esempio, in questa versione di MARS possono girare solo due programmi contemporaneamente. Sarebbe difficile far eseguire più programmi? E che dire di una nuova istruzione di REDCODE che permetta a un programma in funzione di avviare un altro programma, che ha copiato in un'area libera del *nucleo*, aumentando così le possibilità che almeno un programma della «squadra» sopravviva alla battaglia?

L'insieme delle istruzioni di REDCODE presentato è semplice. Quanti fra voi hanno accesso a un calcolatore più grande possono provare a sperimentare nuovi insiemi di istruzioni e nuovi modi di indirizzamento, magari rendendo REDCODE più simile a un vero linguaggio d'assemblatore. Istruzioni che proteggano un programma più grande da un programma piccolo, difficile da battere, favorirebbero l'innalzamento della Guerra dei nuclei a un livello più alto e più interessante.

Linee guida per il cannone ad alianti

di Stephen Wolfram

In questo volume sono descritte alcune fra le ricerche che da tempo conduco sugli automi cellulari unidimensionali. Vi sono alcuni problemi interessanti, relativi a questi sistemi, che forse siete in grado di aiutarmi a risolvere. Sono problemi che non si basano su conoscenze matematiche raffinate, ma probabilmente per fare qualche progresso sono necessari molto lavoro e qualche idea brillante. Queste note definiscono alcuni dei problemi, descrivono quel che sono riuscito a stabilire finora e suggeriscono alcune impostazioni che si potrebbero seguire per cercare di risolverli.

Il tema centrale è stabilire quali tipi di strutture possono presentarsi in certi semplici automi cellulari unidimensionali. Anche se le loro regole fondamentali sono molto semplici, il comportamento complessivo di tali automi cellulari sembra molto complicato. Vorrei poter caratterizzare proprio la loro complessità identificando le strutture e i processi che possono aver luogo in essi.

Molti automi cellulari hanno configurazioni spazio-temporali che sembrano molto caotiche e in cui non è facile identificare strutture definite e persistenti. Vi sono però alcuni automi cellulari, che ho chiamato «classe 4», in cui si presentano strutture definite e localizzate. Questi automi cellulari sono relativamente rari: costituiscono solo il 5 per cento circa delle possibilità (con $k < 5$ e $r < 2$). Vi sono vari tipi di strutture che si possono cercare:

Strutture periodiche. Consistono di configurazioni localizzate che rimangono immutate nel tempo, oppure seguono un ciclo, ritornando periodicamente alla medesima forma. Di solito queste e altre strutture si presentano su uno sfondo di zeri; ma per talune regole lo sfondo può avere un altro stato, o magari essere a sua volta una configurazione periodica «a tappezzeria».

Strutture che si propagano o «alianti». Sono configurazioni localizzate che si spostano sistematicamente nel tempo attraverso il reticolo dell'automa cellulare. Possono avere molte velocità diverse.

Cannoni ad alianti. Secondo la definizione più rigida, sono configurazioni di dimensioni finite che emettono periodicamente alianti. Considerando anche gli alianti, questi oggetti danno origine a configurazioni la cui dimensione totale aumenta sistematicamente. In generale si possono cercare negli automi cellulari di classe 4 strutture le cui dimensioni totali aumentano in questo modo, non necessariamente solo per emissione di alianti. (Si noti tuttavia che negli automi cellulari di classe 3 queste configurazioni sono comuni: il problema è interessante solo per gli automi cellulari di classe 4, come quelli elencati sotto.)

Strutture che si autoriproducono. Sarebbero semplici idealizzazioni matematiche dei sistemi viventi. Una configurazione finita in un automa cellulare di classe 4 crescerebbe nel tempo e alla fine fornirebbe una copia di se stessa. (Si possono verificare fenomeni banali di autoriproduzione in regole che obbediscono a un principio di «sovrapposizione lineare», secondo il quale le configurazioni prodotte da un particolare stato iniziale corrispondono a semplici sovrapposizioni di configurazioni prodotte, poniamo, con un unico seme cellulare.)

Ma alla fine dei conti il problema più interessante, probabilmente, è stabilire se certi semplici automi cellulari unidimensionali sono capaci di «calcolo universale», in modo da emulare il funzionamento di qualunque calcolatore. Esiste un automa cellulare unidimensionale assai complicato, con 18 stati per posizione ($k = 18$), che è stato costruito specificamente per imitare una macchina di Turing universale (un tipico modello matematico di calcolatore), ed è quindi capace di calcolo universale. Da osservazioni empiriche, però, io penso proprio che anche altri automi cellulari molto più semplici, come quelli elencati sotto, siano capaci di calcolo universale. In nessun caso, comunque, esiste una dimostrazione.

Si può provare che un automa cellulare ha la capacità di calcolo universale dimostrando che è in grado di imitare alcuni altri sistemi che già sappiamo essere calcolatori universali. Per un automa cellulare di questo tipo deve essere possibile trovare uno stato iniziale particolare che evolve proprio come uno qualunque degli stati del calcolatore universale noto.

Sono noti vari tipi di calcolatori universali. Una buona introduzione ad alcuni di essi è il libro di F.S. Beckmann intitolato *Mathematical Foundations of Programming*, pubblicato nel 1981 dalla Addison-Wesley. Si può ridurre al minimo la quantità di lavoro necessaria per dimostrare che un automa cellulare è universale dal punto di vista del calcolo scegliendo di verificare l'equivalenza con il più simile fra i sistemi che già sono noti come calcolatori universali.

Una possibilità può essere la dimostrazione della sua equivalenza con una macchina di Turing. Questa è costituita da un nastro (che svolge il ruolo di memoria) diviso in riquadri, ciascuno dei quali contiene un simbolo tratto da un repertorio prefissato, insieme con una «testina» (la CPU, l'unità centrale di elaborazione) che può spostarsi avanti e indietro lungo il nastro, leggendo e scrivendo simboli, secondo un repertorio costante di regole interne. Penso che la macchina di Turing universale più semplice fra quelle note abbia un repertorio di quattro simboli che possono comparire sul nastro e una testina con sette stati interni. In un automa cellulare i simboli sul nastro dovrebbero essere rappresentati da strutture separate stabili (o forse periodiche). La testina sarebbe rappresentata da una struttura costituita da una complessa successione di valori di posizione, che ha opportune interazioni con le strutture dei simboli sul nastro. La trascrizione più diretta della più semplice fra le macchine di Turing note dà l'automa cellulare con 18 stati già citato. Se però si fa in modo che i simboli sul nastro e la testina siano rappresentati da strutture complesse, anziché da singoli valori posizionali, si dovrebbe poter imitare questa macchina di Turing con un automa cellulare più semplice. In effetti, se l'automa cellulare deve essere un calcolatore universale, una simile equivalenza deve esistere sempre, ma la codifica necessaria potrebbe rivelarsi estremamente complicata.

Una seconda possibilità, forse anche migliore, è quella di dimostrare l'equivalenza rispetto a un calcolatore digitale standard. Questa è l'impostazione che è stata usata per dimostrare che l'automa cellulare bidimensionale noto come «Vita» o «Gioco della vita» ha la caratteristica dell'universalità rispetto al calcolo. Si dovrebbe poter seguire molto da vicino lo schema di questa dimostrazione. È necessario identificare nell'automa cellulare oggetti che agiscono come conduttori (che trasportano segnali), memorie, porte logiche e probabilmente un orologio. Molti di questi oggetti sono stati identificati per le regole discusse più avanti, ma ne sono necessari altri. Le memorie corrispondono a strutture periodiche isolate. I segnali vengono propagati da alianti, che si spostano periodicamente nell'automa cellulare. Gli alianti devono interagire fra loro, e con le memorie, per realizzare varie funzioni logiche (come «and», «nand» ecc.). Una volta date «porte» corrispondenti a varie funzioni logiche, si deve scoprire come debbono essere fissate le condizioni iniziali perché l'automa cellulare emuli un circuito digitale arbitrario. Probabilmente saranno necessari anche cannoni ad alianti che fungano da sorgenti di segnali e orologi sincronizzatori.

Per ulteriori informazioni su questi problemi, e in generale sugli automi cellulari, potete leggere qualcuno degli articoli pubblicati da me. Due sono stati scritti per un pubblico abbastanza ampio: *Cellular automata* in «Los Alamos Science» primavera 1983 (può essere ottenuto dal Los Alamos National Laboratory, Los Alamos, NM 87545); *Cellular automata as models of complexity* in «Nature» volume 311, p.419, 4 Ottobre 1984.

Articoli più tecnici sono: *Statistical mechanics of cellular automata* in «Reviews of Modern Physics» volume 55, p. 601, luglio 1983; *Universality and complexity in cellular automata* in «Physica D», volume 10, p. 1, 1984. Quest'ultimo articolo è apparso anche in un libro intitolato *Cellular Automata*, a cura di D. Farmer, T. Toffoli e S. Wolfram, pubblicato nel 1984 dalla North Holland/Elsevier.

Un articolo un po' più tecnico, che descrive in particolare alcuni fra gli algoritmi discussi più avanti in queste note è *Computation theory of cellular automata*, in «Communications in Mathematical Fhysics», volume 96, p. 15, 1984.

Le regole degli automi cellulari

Ecco alcune regole per automi cellulari di classe 4 nelle quali cercare vari tipi di strutture. Personalmente sono dell'opinione che ciascuna di queste regole sia caratterizzata dalla proprietà dell'universalità rispetto al calcolo, ma sarebbe molto interessante riuscire a dimostrarlo.

1. $k = 3$, $r=1$, codice di regola totalistica 357: sono state trovate varie strutture periodiche e di recente io ho trovato un aliante molto complicato.

2. $k = 2$, $r = 2$, codice di regola totalistica 20: sono note strutture periodiche e strutture che si propagano, ma non è stato mai trovato alcun cannone ad alianti e non sono state studiate le interazioni fra le strutture.

3. $k = 3$, $r = 1$, codice di regola totalistica 792: sono note strutture periodiche e strutture che si propagano, ma non è stato mai trovato alcun cannone ad alianti.

4. $k = 2$, $r = 3$, codice di regola totalistica 88 (regola di Park): sono state trovate strutture periodiche e strutture che si propagano, e anche un cannone ad alianti. Ora debbono essere assemblate per realizzare un calcolatore universale.

5. $k = 2$, $r = 1$, numero di regola 193: sono state trovate strutture periodiche e strutture che si propagano, e sono state studiate alcune fra le loro interazioni. Nelle altre regole, lo «sfondo» consiste di cellule nello stato 0. Qui vi è uno sfondo «a tappezzeria» alquanto elaborato, con periodo 14 orizzontalmente e 7 verticalmente.

Note

Tutti i fenomeni dovrebbero essere studiati su reticoli abbastanza ampi perché gli effetti ai bordi siano irrilevanti.

I valori di k dati specificano il numero degli stati possibili (etichettati, poniamo, da 0 a $k - 1$) nell'automa cellulare. Stati diversi possono essere rappresentati da colori o caratteri diversi. I valori di r danno il «dominio» della regola dell'automa cellulare. Quando $r = 1$, il nuovo stato di una cellula particolare dipende solo dalle cellule immediatamente adiacenti alla sinistra e alla destra. Quando $r = 2$, dipende dalle due cellule più vicine per parte, e via dicendo.

I codici di regola «totalistica» usati nei casi 1-3 sono definiti nell'articolo *Costruire calcolatori a una sola dimensione getta luce su fenomeni irriducibilmente complessi* a pagina 90. Per esempio, l'automa cellulare $k = 2$, $r = 2$ con codice 20 (caso 2 sopra) ha la regola seguente. In primo luogo, si trovi la somma numerica degli stati di una posizione e delle posizioni immediatamente adiacenti a sinistra e a destra. Se questa somma (il cui valore massimo è 5) è esattamente 2 o 4, allora la nuova cellula avrà stato 1; altrimenti avrà stato 0. Bisogna ricordare che la somma va calcolata sui vecchi stati delle cellule; i nuovi stati appena calcolati debbono essere tenuti ben distinti. La regola per il caso 5 è di tipo diverso. Innanzitutto si trovi l'equivalente binario di 193: è 11000001. Allora l'i-esima cifra (contando a partire da zero sulla destra) dà il nuovo stato di una cellula quando il numero binario formato dai tre vecchi stati delle cellule è esattamente i. Così se le tre cellule indicano 000, 110 o 111, la nuova cellula ha stato 1; altrimenti ha stato 0.

Qualche altra regola

Ecco alcune altre regole per automi cellulari di classe 4:

$k = 5$, $r = 1$, codice totalistico 53955: sfondo di 1.
$k = 5$, $r = 1$, codice totalistico 522809355; sfondo con periodo 2.
$k = 5$, $r = 1$, codice totalistico 55135.
$k = 4$, $r = 1$, codice totalistico 1004600.
$k = 2$, $r = 2$, codice totalistico 52.

Procedure di ricerca

Conosco due metodi fondamentali che possono essere usati per la ricerca di strutture negli automi cellulari unidimensionali. Il primo consiste semplicemente nel controllare ogni possibile configurazione iniziale e osservarne il destino. Il secondo consiste nell'usare un algoritmo (descritto sotto) che può trovare sistematicamente tutte le strutture con un periodo particolare in un dato automa cellulare.

Non è molto facile trovare strutture complesse negli automi cellulari. I metodi sistematici tendono ad assorbire moltissimo tempo di calcolo. Ciascuna delle tabelle riportate più avanti ha richiesto molte ore di tempo macchina su un calcolatore ragionevolmente veloce (per lo più ho usato un Ridge 32, che per velocità è paragonabile a un grosso VAX). Per esempio, ci sono voluti circa due giorni di tempo macchina per trovare l'aliante nella prima tabella. Però ho fatto fare al calcolatore tutto il lavoro; penso che se avessi dedicato abbastanza tempo allo studio delle configurazioni prodotte dall'automa cellulare, sarei riuscito da solo a percorrere molta strada nella costruzione di strutture interessanti, senza dover lasciare al calcolatore il compito di ricercare attraverso tutte le possibilità. Penso che la cosa migliore sia probabilmente combinare la sperimentazione con un po' di riflessione e un po' di ricerca sistematica al calcolatore.

Quando si effettua una ricerca esaustiva di tutte le configurazioni iniziali possibili, poniamo fino a una certa dimensione data, le stesse strutture verranno prodotte molte volte. Si può stabilire che una struttura ben definita è stata prodotta osservando se le configurazioni prodotte sono periodiche nel tempo. Poi si deve scoprire se la struttura è già stata prodotta in precedenza. Può essere un po' tortuoso, perché la si può individuare in punti diversi del suo ciclo in occasioni diverse. Inoltre bisogna tener conto degli alianti, che si ripetono, ma spostati. Il mio programma osserva sempre solo una «finestra» sull'automa cellulare i cui bordi sono posti a distanza di sole r cellule dai bordi delle configurazioni prodotte e che si sposta quando si sposta la configurazione. Un altro problema è che talune configurazioni iniziali possono evolvere in molte strutture distinte, ciascuna delle quali è stata già vista in precedenza. Bisogna far sì che il programma sia in grado di riconoscere queste strutture. Un caso un po' strano comporta strutture composte con alianti che vanno in direzioni opposte, o alianti che si spostano insieme con strutture periodiche. In questi casi, la configurazione non diventa mai periodica nel suo complesso, anche se diventano periodiche le sue parti.

La procedura testé descritta può essere usata in linea di principio per trovare tutte le strutture che un particolare automa cellulare può generare. Si può controllare ogni possibile configurazione iniziale, sistematicamente, oppure si possono provare configurazioni iniziali scelte casualmente.

Una procedura alternativa consiste nell'utilizzare un algoritmo sistematico che trovi tutte le strutture possibili con un periodo particolare in un dato automa cellulare. Ovviamente, come mostrano le tabelle, alcune fra queste strutture possono avere un periodo molto lungo, e l'algoritmo diventa del tutto privo di praticità al di là di periodi molto brevi. Descriverò innanzitutto come fa quest'algoritmo a trovare configurazioni che sono stabili sotto una regola di automa cellulare e di conseguenza restano immutate da un istante temporale al successivo.

1. Scriviamo lo stato ottenuto applicando la regola dell'automa cellulare a ciascun blocco possibile di $2r + 1$ stati della cellula. Poi scartiamo i blocchi in cui lo stato della cellula centrale si modifica. Solo i blocchi restanti possono apparire nei blocchi restanti. Ora si deve semplicemente trovare quali configurazioni (se ve ne sono) possono essere ottenute concatenando fra loro questi blocchi permessi, con i blocchi che si sovrappongono per 2 posizioni. Per maggiore efficienza, a questo punto è bene creare una tabella che specifichi quale blocco può seguire qualche altro blocco in una configurazione. (Tabelle del genere sono equivalenti ai cosiddetti grafi «di de Bruijn».) Così, per esempio, 101 può essere seguito solo da 010 e 011 (anche se uno o ambedue possono non essere presenti nell'insieme di blocchi per il quale la cellula centrale viene messa in corrispondenza con se stessa).

2. Ora costruiamo un «albero» di tutte le possibili concatenazioni di blocchi. Assumiamo che nello «sfondo» tutte le cellule abbiano stato 0. Poi cominciamo con il blocco che rappresenta lo sfondo (per esempio 000 per una regola $r = 1$). Poniamo questo blocco come radice di un albero. Poi costruiamo rami verso nodi corrispondenti ai blocchi che possono seguire il blocco attuale. Continuiamo questa procedura finché (a) non vi sono blocchi permessi che possano

seguire il blocco attuale, o (*b*) si raggiunge di nuovo il blocco di sfondo. Nel caso (*b*), la successione di blocchi sul cammino che inizia alla radice corrisponde a una configurazione periodica nell'automa cellulare (la configurazione effettiva può essere letta dalla successione degli stati della cellula centrale nei blocchi).

3. Per cercare strutture con periodo p maggiore di 1, basta formare blocchi di lunghezza $2r + 1$ che rappresentano l'effetto di p applicazioni della regola dell'automa cellulare, e poi usare l'algoritmo appena delineato. Questo algoritmo può essere utilizzato anche per la ricerca di alianti con una velocità predeterminata: basta far sì che i blocchi diano i risultati della regola dell'automa cellulare spostati della quantità opportuna.

Tabelle di strutture

Strutture con periodi seguiti da una S o da una D sono alianti che si muovono, rispettivamente, verso sinistra e verso destra. I cannoni ad alianti sono indicati con una C.

Per ciascuna struttura è dato un codice per la configurazione iniziale più breve (il «seme») che la genera. Il codice è costituito da una successione di stati cellulari per la configurazione, trattati come cifre di un numero in base k, e dati come numero decimale. (Così, per esempio, la configurazione 2 con $k = 3$ è $102 = 3^3 + 2$.)

periodo	seme
48	28
19	7795
19	8083
26	1706588
41S	4803890
41D	12269314

Strutture identificate per la regola $k = 3$, $r = 1$ con codice 357. (Ricerca condotta fino alla dimensione 14.)

periodo	seme
2	151
9D	187
1	189
22	195
9S	221
1D	635
1S	889
38	125231
4	595703
4	610999
4	624623

Strutture identificate per la regola $k = 2$, $r = 2$ con codice 20. (Ricerca condotta fino alla dimensione 21.)

periodo	seme
1	4
20	5
3D	101
3S	113
16	1625
3S	4187
3D	4561
3D	5252
3S	5540
12	12031
32S	35996
32D	40424
6D	141872
6S	148424

Strutture identificate per la regola $k = 3$, $r = 1$ con codice 792. (Ricerca condotta fino alla dimensione 11.)

periodo	seme
3	7
1D	207
1S	243
2	1631
2	3295
1	3579
3D	6591
3S	7135
238C	7167
34	1704415
3S	57307736539103
3D	69268037164555

Strutture identificate per la regola $k = 2$, $r = 3$ con codice 88 da James Park della Princeton University. (Ora sono note tutte le possibili strutture che hanno una periodo massimo 3; quelle non elencate nella tabella hanno la forma 1101000001111011111111101000 (110111111011101111000)*1110111111111011111 dove la successione identificata dall'asterisco può essere omessa o ripetuta un numero qualunque di volte.

Algoritmi realizzati e proposti per la soluzione del problema del pangramma di Lee Sallows

di A.K. Dewdney

Il problema:

«Scommetto 10 ghinee che nessuno riuscirà a trovare una soluzione che si autoenumera (o la dimostrazione della sua non esistenza) per l'enunciato che inizia "This computer-generated pangram contains... and..." nell'arco dei prossimi 10 anni.» Lee Sallows.

La soluzione:

This computer-generated pangram contains
six a's, one b, three c's, three d's,
thirty-seven e's, six f's, three g's, nine
h's, twelwe i's, one j, one k, two l's,
three m's, twenty-two n's, thirteen o's,
three p's, one q, fourteen r's, twenty-
nine s's, twenty-four t's, five u's, six
v's, seven w's, four x's, five y's and one z.

Introduzione

Nel momento in cui scrivo queste note (dicembre 1984) ci sono cinque pretendenti al premio di Lee Sallows. A parte i quattro risolutori citati a pagina 145, hanno trovato una soluzione anche Michael J. Gayle e James M. Mittan di Ft. Wayne, Indiana. Ora il fatto che le cinque soluzioni siano identiche mi sorprende meno di quanto mi sia successo dopo aver visto le prime quattro: forse, all'interno del dominio preso in considerazione esiste una sola soluzione.

Nella tabella che segue ho inserito tutte le informazioni pertinenti relative alle cinque soluzioni. È stupefacente la varietà di macchine, linguaggi di programmazione e tempi di risoluzione.

NOME	MACCHINA	LINGUAGGIO	TEMPO IMPIEGATO	DATA
John R. Letaw	VAX 11/780	FORTRAN	5 minuti	20/9/84
Lawrence G. Tesler	Apple Lisa	PASCAL	2 ore	23/9/84
E.M.	VAX 11/780	FORTRAN	40 ore	8/10/84
William B. Lipp	IBM PC	PASCAL	x ore*	18/10/84
Gayle & Mittan	IBM 3801	COBOL	12 minuti	19/11/84

* Lipp ha lasciato in funzione la sua macchina per un weekend e ha trovato la soluzione sulla stampante al suo ritorno.

Gayle e Mittan non hanno inviato documentazione in tempo per l'inclusione in queste note, ma negli altri quattro casi ho estratto parti delle lettere e dei documenti in modo che i lettori interessati possano avere qualche idea sul modo in cui ciascuno è riuscito a ottenere il risultato. Gli algoritmi usati vanno da una impostazione da «risalita» fino a una potatura massiccia.

Oltre ai cinque risolutori, due lettori, Hans Buchwald di Copenaghen e Robert Wolfson di Atlanta, Georgia, hanno proposto degli algoritmi. Le loro idee sono riportate alla fine di queste note, nell'eventualità che in futuro qualche «pangrammaniaco» le trovi utili.

Algoritmi implementati

John R. Letaw:

«Accludo particolari sul programma. Il VAX 11/780 dello US Naval Research Laboratory (dove lavoro come fisico, per ricerche sui raggi cosmici) ha impiegato meno di cinque minuti per risolvere il problema. Il numero di occorrenze delle lettere a, b, c, d, j, k, m, p, q e z è determinato dal numero di volte che appaiono nella locuzione "THIS COMPUTER GENERATED PANGRAM CONTAINS AND", a cui si deve aggiungere un'unità per l'occorrenza di ciascuna lettera nell'elenco. Il "nucleo" delle presenze di e, f, g, h, i, l, n, o, r, s, t, u, v, w, x, y è costituito dalle stesse due fonti precedenti; in più bisogna aggiungere la loro frequenza nelle parole che indicano il numero di occorrenze delle lettere che non appartengono alla "cornice".

Se gli indici vanno da 1 a 16 (uno per ciascuna lettera della "cornice"), allora il nucleo è un vettore

$$C(j) = (18,1,3,6,4,1,11,8,8,9,9,2,1,1,2,1)$$

e il numero finale di ciascuna è

$$N(j) = (37,6,3,9,12,2,22,13,14,29,24,5,6,7,4,5)$$

Vale la relazione

$$N(j) = C(j) + \sum_k D[N(k),j]$$

dove $D[\ ]$ è la frequenza dell'i-esima lettera della cornice in $N(k)$. Il problema viene risolto per iterazione, con la costruzione di una nuova stima di N (chiamata F') a partire da una stima precedente di N (chiamata F) che inizialmente è posta uguale al nucleo. Le equazioni sono:

$$A(j) = C(j) + \sum_k D[F(k), j]$$

$$F'(j) = \mathrm{INT}[a * Z(j) + (1-a) * F(j) + 0.5]$$

dove a è un numero casuale compreso fra 0 e 1. Queste equazioni sono soddisfatte identicamente da N e da null'altro, quando vengono iterate. La presenza di F nella seconda equazione riduce le oscillazioni di instabilità. Il numero casuale elimina alcuni punti stabili accidentali.

Questo metodo è molto efficace per catturare soluzioni approssimate al problema del pangramma e passare a nuove soluzioni approssimate. Quando la trova, si fissa sulla soluzione effettiva. Nell'esempio sopra riportato sono state tentate 84 000 soluzioni. Si tratterebbe di un compito affrontabile su un microcalcolatore, lasciandolo attivo per un paio di giorni. In effetti, ho sviluppato il programma su un microcalcolatore. Uno svantaggio di questo metodo sta nel fatto che è completamente indeterminato. Non offre alcuna garanzia di poter identificare una soluzione, se ne esiste almeno una, e non offre alcun metodo per dimostrare l'impossibilità di una soluzione.

Cercando una prova ulteriore del metodo ho trovato un pangramma che si autodocumenta in modo più completo:

This computer-generated pangram containing eight a's, two b's, three c's, four d's, thirty-six e's, eight f's, seven g's, twelwe h's, fourteen i's, two j's, one k, three l's, three m's, twenty-two n's, sixteen o's, three p's, one q, fifteen r's, thirty-one s's, twenty-eight t's, seven u's, four v's, nine w's, four x's, six y's, and one z was found by John R. Letaw.

Questo pangramma ha richiesto meno di 20 000 iterazioni.»

Lawrence G. Tesler:

«Il pangramma è stato trovato da un programma in PASCAL che ho scritto e mandato in esecuzione su un calcolatore Apple Lisa. Il programma ha valutato le possibilità di circa 250 000 000 000 frasi in due ore. Gli intervalli del dominio di ricerca erano:

e=35-42, f=5-11, g=3-6, h=7-11, i=9-15, l=1-5, n=17-24, o=11-15, r=10-15, s=25-31, t=20-25, u=2-6, v=6-10, w=4-9, x=2-5, y=4-5

Con questi intervalli di ricerca vi sono 474 163 200 000 frasi possibili.

Il programma finale ha eliminato oltre metà dei candidati prima di inciampare in una soluzione. La velocità effettiva di eliminazione era di circa 38 000 000 di combinazioni al secondo. L'unità di elaborazione Motorola 68 000 da 5 MHz del Lisa è veloce, ma non così tanto. Il programma ha controllato esplicitamente un totale di 1 200 914 frasi (circa 180 al secondo) e ha applicato una regola di potatura (descritta in seguito) per eliminare una media di 210 000 frasi per ognuna controllata.

Per capire il programma, è utile fare riferimento all'illustrazione di pagina 68. Definiamo la funzione $Cj = z(Ci)$ come l'operazione definita dall'algoritmo illustrato; prende cioè la "combinazione attuale" Ci mostrata nella colonna più a destra dell'immagine e produce la "combinazione somma di colonne" Cj mostrata nella riga inferiore.

Il programma applica la funzione z alla prima combinazione $C1$ negli intervalli prescelti. Se $x(C1) = C1$, allora il problema è stato risolto. Altrimenti, il normale algoritmo dell'odometro incrementa l'ultimo termine e avvia l'iterazione sulla combinazione successiva $C2$. Invece, il nostro programma applica la regola della potatura.

Questa regola di potatura si può spiegare meglio con un esempio. Nell'illustrazione citata, perché la combinazione sia giusta mancano 8 T. Se il valore delle Y può variare fra 1 e 10, allora modificando il valore di Y da 5 a qualche altro valore si potrebbe ottenere al più un'altra T (*two, three, eight* e *ten* hanno una T, mentre *five* non ne ha alcuna). Con un ragionamento analogo un diverso numero di U (o di V, di W o di X) potrebbe fornire al più un'altra T. Nel migliore dei casi possiamo aggiungere cinque T provando tutte le migliaia di combinazioni di U, V, W, X e Y, il che ci lascia ancora con tre T mancanti. Sarebbe del tutto infruttuoso tentare qualunque altra combinazione di queste cinque lettere con il valore attuale di T, e pertanto possiamo provare a incrementare T. Ma in questo modo non facciamo altro che peggiorare le cose: il nostro «buco» aumenterebbe, anziché diminuire. Pertanto riportiamo T al suo valore minimo e procediamo a incrementare S.

La regola di potatura, con gli intervalli visti prima, ha dato un aumento di velocità di 210 000 volte: senza di essa, il programma avrebbe potuto girare 100 anni per trovare una soluzione.

Ho scelto questi intervalli analizzando alcune "quasi soluzioni" incontrate nel corso di esecuzioni iniziali al calcolatore, di carattere esplorativo.»

E. M.:

«Cominciamo la discussione dell'algoritmo utilizzato notando una condizione, semplice e ovvia, che qualunque pangramma deve soddisfare: il numero totale delle lettere nella frase deve essere pari alla somma dei valori, espressi in lettere, delle 10 lettere la cui frequenza è nota e delle 16 la cui frequenza è ignota.

Il conteggio di tutte le lettere nella frase può essere espresso sotto forma di somma:

$$Ctot = Cbase + [\text{somma sulle 16 incognite: } C(Vi)] \qquad (1)$$

dove

- $Ctot$ è il conteggio di tutte le lettere nella frase.
- $Cbase$ è il conteggio di tutte le lettere nella frase base.
- Vi è il coefficiente di valore per la lettera i (cioè il valore numerico rappresentato dalla parola che sostituisce un "?").
- $C(Vi)$ è il conteggio delle lettere nella parola per il valore Vi. (Per esempio, C(6)=4, perché la parola corrispondente al valore 6 è "sixs" e ha 4 lettere.)

La somma dei 26 valori formulati in lettere può essere espressa come:

$$Vtot = Vbase + [\text{somma sulle 16 incognite: } Vi] \qquad (2)$$

dove

- $Vtot$ è il totale di tutti i 26 coefficienti di valore.
- $Vbase$ è il totale dei 10 coefficienti di valore noti.
- Vi è il coefficiente di valore per la lettera i (come sopra).

Se si impone che $Vtot = Ctot$, si ha:

$$Cbase - Vbase = [\text{somma sulle 16 incognite: } (Vi - C(Vi))] \qquad (3)$$

Nell'equazione (3) $Cbase$ e $Vbase$ sono noti all'inizio, e $C(Vi)$ è noto se è noto Vi. Così, se generiamo tutti i possibili insiemi [Vi] di 16 valori incogniti che sono soluzione di (3), possiamo controllare ciascuno per vedere se è una soluzione per un pangramma. Se abbiamo un insieme [Vi] (di 16 coefficienti), possiamo controllarlo per vedere se è una soluzione possibile per un pangramma contando l'incidenza di ciascuna delle 16 lettere incognite nella frase corrispondente. Se l'insieme di 16 numeri che ne risulta corrisponde ai 16 valori di [Vi], abbiamo una soluzione. È da notare che non è necessario sapere subito quale Vi vada associato a quale fra le 16 lettere incognite, se [Vi] rappresenta effettivamente una soluzione (il test per verificare se è una soluzione fornirà quell'informazione come sottoprodotto).

Riscriviamo (3) nella forma:

$$Cbase - Vbase = [\text{somma sulle 16 incognite: } Ni] \qquad (3')$$

dove

$$Ni = Vi - C(Vi)$$

Ni è il "conteggio netto" associato al valore Vi. Per esempio, se $Vi = 10$, allora $Ni = 10 - 4 = 6$ (perché «tens» ha 4 lettere). Se si prepara una tabella con il "valore", la "parola", il "conteggio" e il "conteggio netto" per i valori da 1 a 20, si può notare che in molti casi un dato conteggio netto è associato con due valori diversi, e che in un caso un conteggio netto (4) è associato con 3 valore diversi (9, 11, 13).

L'equazione (3') fa pensare a una strategia possibile: generare tutti gli insiemi [Ni] di 16 valori netti che sono soluzioni di (3'), poi controllare ciascuno per vedere se è una soluzione del pangramma. È abbastanza immediato generare efficientemente tutti gli insiemi [Ni] coerenti con (3'), ma quale controllo si potrebbe applicare a un particolare insieme [Ni] per vedere se rappresenta una possibile soluzione del pangramma? La complicazione qui è che molti dei possibili valori netti corrispondono a più di un valore. Per esempio, un valore netto 2 corrisponde sia a 6 sia a 8. Così un singolo insieme di valori netti [Ni] in generale corrisponderà a molti possibili insiemi di valori [Vi]. Un metodo diretto sarebbe quello di generare e controllare (come descritto sopra) tutti i [Vi] corrispondenti a ciascun [Ni]. Questa procedura, però, richiede molto tempo. Quello che vogliamo è un controllo efficiente che possa rifiutare un particolare [Ni] come possibile soluzione del pangramma senza controllare tutti i corrispondenti [Vi].

Ora descriverò il controllo principale che ho utilizzato per rifiutare come possibili soluzioni del pangramma la maggior parte degli insiemi [Ni] che sono soluzione di (3'). Il controllo (che chiamo TEST-1) è costituito da questi passi:

1) Generare ciascun insieme [Ni] che soddisfa (3') con il requisito ulteriore che

$$N1 \leq N2 \leq N3 \leq \ldots \leq N16.$$

(Questo requisito di ordinamento è motivato solo da ragioni di comodità ed efficienza e non è parte essenziale dell'argomentazione.)

2) Un dato valore netto Ni corrisponde a uno o più valori; siano VMINi il più piccolo e VMAXi il più grande di tali valori, per ciascun Ni. Poi definiamo

(VVMINi) = i 16 (VMINi), in ordine crescente,
e (VVMAXi) = i 16 (VMAXi), in ordine crescente.

3) Analogamente, anche un dato valore netto Ni corrisponde a una o più parole (una per ciascun valore corrispondente). Considerando tutte le scelte possibili per queste parole (su tutti i 16 Ni in [Ni]), possiamo definire per ciascuna delle 16 lettere incognite:

- CMIN(L) = la minima incidenza possibile della lettera incognita L su tutte le frasi corrispondenti a [Ni],
- CMAX(L) = la massima incidenza possibile della lettera incognita L su tutte le frasi corrispondenti a [Ni].

Si noti che CMIN (L) e CMAX(L) comprendono l'incidenza di L tanto nella frase base quanto nei 16 valori incogniti

(CCMINi) = i 16 (CMIN(L)), in ordine crescente,
(CCMAXi) = i 16 (CMAX(L)), in ordine crescente.

4) Ora supponiamo che esista una soluzione per il pangramma, [Vi], che corrisponde all'insieme [Ni] considerato, e sia [VVi] l'insieme [Vi] (ordinato) in ordine crescente. Allora deve essere vero che

CCMINi $\leqslant$ *VVi* $\leqslant$ CCMAXi, per tutti gli *i* = 1,..., 16
e VVMINi $\leqslant$ *VVi* $\leqslant$ VVMAXi, per tutti gli *i* = 1,..., 16

Questa conclusione può non risultare immediatamente ovvia, ma l'eventuale ulteriore chiarimento necessario può venire meglio da un po' di riflessione da parte del lettore che non da altre parole dell'autore.

5) Ora possiamo formulare il controllo (TEST-1):

Una soluzione di (3'), [*Ni*], può essere rifiutata come possibile soluzione del pangramma se esiste qualche *i* (i = 1,..., 16), per il quale o CCMINi > VVMAXi, oppure CCMAXi < VVMINi.
Questo controllo segue direttamente dal punto 3).

Il controllo che ho appena descritto (TEST-1) è il controllo chiave usato nel mio programma, che cerca in modo esaustivo le soluzioni al pangramma. Rifiuta la maggior parte degli insiemi [*Ni*] coerenti con (3'), e quelli che rimangono possono essere controllati con il metodo diretto di generare tutti i [*Vi*] corrispondenti e di controllare ciascuno nel modo descritto precedentemente. Ora vorrei citare brevemente due altri controlli che il programma usa insieme con TEST-1, controlli che sono stati aggiunti al fine di ridurre il tempo complessivo di esecuzione del programma. Il primo (TEST-0) viene applicato prima di TEST-1 mentre il secondo, TEST-2, viene applicato dopo TEST-1 (se necessario), ma prima del controllo esaustivo di tutti i [*Vi*].

TEST-0

Si può notare che l'equazione (4a) del TEST-1 può essere applicata come controllo a un insieme parziale [*Ni*], nel quale sono stati selezionati meno di 16 *Ni*. Se questo controllo produce un rifiuto, allora in generale rifiuterà uno o più insiemi [*Ni*] (*i* = 1,...,16) che sono soluzioni di (3') ma che non possono includere alcuna soluzione per il pangramma.

TEST-2

Per un particolare [*Ni*] soluzione di (3'), può essere possibile identificare qualche valore *V* che deve essere parte di qualunque soluzione [*Vi*] che corrisponde a [*Ni*]. Qualunque lettera la cui incidenza è invariante rispetto a tutti i possibili [*Vi*] coerenti con [*Ni*] identifica un valore *V* di questo tipo. Una lettera del genere avrebbe CMIN(L) = CMAX(L), per usare la terminologia definita nel TEST-1. Inoltre, qualunque *i* (*i* = 1,..., 16) in cui o CCMAXi = VVMINi oppure CCMINi = VVMAXi identifica un valore *V* che deve essere presente. Nel TEST-2 si richiede che i valori netti *N* corrispondenti a tutti questi valori *V* necessari siano presenti in [*Ni*], altrimenti [*Ni*] viene rifiutato perché non può contenere una soluzione del pangramma.

L'algoritmo in pratica

Per un dato problema, dobbiamo specificare l'ampiezza dell'intervallo di conteggi netti su cui vogliamo effettuare la ricerca. Il massimo conteggio netto possibile in genere si ha se si assume che tutte le 16 lettere incognite abbiano l'incidenza massima della lettera "e", che è pari a 4. Se prendiamo in considerazione un intervallo di conteggi netti che consenta valori grandi quanto l'incidenza della "e" nella frase base più 4*16, allora quell'intervallo contiene di sicuro tutte le possibili soluzioni del pangramma. Abbiamo scelto un limite superiore così elevato perché con questo algoritmo la penalizzazione, in termini di aumento del tempo di calcolo, è leggera.

Prestazioni sul problema "standard"

Il programma ha effettuato una ricerca esaustiva delle soluzioni al problema "standard" dell'articolo a pagina 137 permettendo valori incogniti con limite superiore 71 (7 + 4 * 16). Se dovessimo generare tutti gli [*Ni*] coerenti con (3') per questo problema, ci sarebbero circa 46 milioni di queste soluzioni; in effetti, usando TEST-0, non è necessario generarle tutte, ma solo 14 milioni. 230000 di queste superano TEST-1 e il 57 per cento di quelle che passano TEST-1 superano anche TEST-2. Restano dunque da controllare in forma esaustiva 132000 diversi [*Ni*] (mediante un controllo di tutti i [*Vi*] corrispondenti). Il tempo di esecuzione totale per questo problema è inferiore alle 10 ore (su un VAX-11/780, con un programma scritto in FORTRAN). Il problema ha due soluzioni del pangramma. Una è data nell'articolo, la seconda è:

This pangram contains four a's, one b, two c's, one d, twentysix e's, six f's, three g's, six h's, eleven i's, one j, one k, two l's, two m's, seventeen n's, fifteen o's, two p's, one q, eight r's, thirty s's, seventeen t's, four u's, four v's, six w's, six x's, three y's one z.

Prestazioni sul problema "delle dieci ghinee"

Il problema "delle dieci ghinee" è stato risolto in maniera analoga dallo stessa programma. In questo caso il tempo di esecuzione è stato maggiore, circa 40 ore.»

William B. Lipp:

«Ho usato un PC IBM con 64 K di memoria. Mi sono servito di un programma in PASCAL, che conservava traccia dei valori "quasi" corretti e cercava nuove possibilità "vicine" a essi. Scartava le possibilità meno approssimate, secondo necessità, in modo da rimanere entro i limiti della memoria disponibile alla macchina. Ho fatto girare in momenti diversi versioni diverse; la versione finale che ha avuto successo ha cominciato a girare la sera di giovedì 18 ottobre. Sono partito per il weekend e ho trovato la risposta ad attendermi, al mio ritorno, domenica 21 ottobre.
La mia misura di "prossimità" era la somma dei quadrati della differenza fra le frequenze espresse e quelle effettive. Il programma non era particolarmente ottimizzato. Generava il controllo effettivo da un elenco di 26 numeri, e contava le lettere nel controllo generato. Credo che esistano altre soluzioni, ma ho scritto il programma in modo che si fermasse dopo averne trovata una.
Questa informazione potrebbe includere ricerche sul problema del commesso viaggiatore, sulla soluzione del problema dei quattro colori e sull'evoluzione dei programmi. Il mio obiettivo originale era quello di generare frasi, ciascuna delle quali effettivamente enumerasse la precedente, e di lasciarlo girare fino a che "convergesse" su un ciclo autoenumerantesi. Il programma non ha funzionato, ma l'idea è rimasta come una delle euristiche nel programma finale. Anche la struttura di dati utilizzata per conservare le risposte meglio approssimate è stata sottoposta a numerose revisioni.»

Algoritmi proposti

Hans Buchwald

«È ovvio che per competere con la macchina di Sallows, che sottopone a controllo una soluzione possibile in un microsecondo, l'algoritmo deve poter scartare la maggior parte dei candidati inutilizzabili con poche istruzioni di macchina.
Prendendo come punto di partenza l'algoritmo descritto nella parte finale dell'articolo a pagina 137, codificando in un linguaggio di alto livello e utilizzando un'impostazione abbastanza ingenua, sono arrivato alla stessa conclusione di Sallows: ogni controllo richiede un tempo dell'ordine di 10000 microsecondi. Si può guadagnare sicuramente un fattore 50 con una codifica in linguaggio di assemblatore e con l'uso di vari "trucchi del mestiere", ma anche così si rimane molto lontani dalle prestazioni della macchina dedicata.
Tuttavia, dopo aver pensato un po' al problema, mi sono venute due idee, che rendono l'algoritmo sufficientemente veloce. La prima è che il controllo viene effettuato solo se la soluzione candidata soddisfa il pangramma modulo due. In questo modo è necessario un solo bit per ciascun conteggio in carattere e, dovendo considerare solo 16 caratteri (finché il conteggio rimane al di sotto di 100), tutti i conteggi stanno comodamente in una parola di 16 bit. La somma dei caratteri indipendenti è semplicemente una operazione di OR esclusivo, disponibile su tutti i calcolatori digitali.
Statisticamente, solo un canditato su 2^{16} supererà questo controllo, poi i candidati sopravvissuti dovranno essere controllati con il "metodo ingenuo". Non ha importanza allora se sono necessari complessivamente anche 10000 microsecondi, perché solo 0,15 microsecondi bastano per esaminare ogni candidato.
La seconda idea è questa: nel passare in rassegna le possibili soluzioni, vengono apportate solo modifiche secondarie agli attributi che determinano se i candidati risolvono o meno il pangramma modulo due. In effetti ciascun controllo di un candidato può essere ridotto a una operazione di OR esclusivo fra un registro e una costante immediata, e al successivo confronto con zero. In questo modo il controllo può essere eseguito in pochi microsecondi, in un tempo cioè abbastanza veloce perché l'algoritmo risulti, sotto il profilo computazionale, efficiente quanto la macchina dedicata.

Al fine di introdurre la notazione necessaria per spiegare la mia soluzione, vorrei prima descrivere in modo formale l'algoritmo introdotto brevemente a pagina 141.

Innanzitutto si assegna un intero a ciascuno dei 16 caratteri che si devono prendere in considerazione, in modo che "e" sia il numero 0, "f" sia 1, "g" sia 2,... e infine "y" sia il numero 15.

Ora possiamo compilare una tabella di profili:

$$P[0..??,\ 0..15]$$

dove (per $0 < i < 100$) la riga, $P[i,..]$, è il profilo del numerale i. Cioè: l'elemento $P[i, j]$ è il numero di volte in cui il carattere j si presenta nel numerale i (in inglese). Analogamente, la riga $P[0,...]$, è il profilo dell'"enunciato scheletro".

Ora una possibile soluzione è data nella forma:

$$C[0..15]$$

dove $C[k]$ è il conteggio del numero k di carattere. Dopodiché possiamo formare il vettore somma:

$$S[j] = P[0, j] + \sum_{i=0}^{15} P[\ C[i], j]$$

Il candidato è una soluzione del pangramma se e solo se:

$$S = C.$$

La prima idea poi era quella di controllare solo che l'equazione fosse soddisfatta modulo due. Vediamo in che modo quest'idea consenta di rendere più compatti i dati e più veloce l'algoritmo.

Innanzitutto si può comprimere la tabella di profili in un vettore:

$$P[0..99]$$

ciascun elemento del quale è una parola di macchina di sedici bit. Sia:

$$Wi$$

l'i-esimo bit della parola W, con 0 come bit meno significativo e 15 come bit più significativo. Il vettore profilo poi viene generato a partire dalla vecchia tabella di profilo mediante la semplice regola:

$$P[j]_i = (P[j,\ i]) \bmod 2.$$

Supponiamo di avere una possibile soluzione $C[\,]$, costituita da un vettore di interi. A partire dalla parola di macchina possiamo generare il vettore C mediante la definizione:

$$C_i = C[i]_0.$$

Questo significa che C registra la parità dei conteggi di ciascuno dei sedici caratteri.

Per ottenere la parità di ciascuno degli elementi nel vettore S (codificato in una parola di macchina S) dobbiamo semplicemente applicare la formula:

$$S = P[0] \oplus \bigoplus_{i=0}^{15} P[C(i)]$$

dove $\oplus$ indica l'addizione modulo due bit per bit, quella che normalmente nel mondo dei computer viene chiamata OR esclusivo.

Se C è una soluzione modulo due si vedrà mediante l'identità $S = C$ o, in modo ancor più comodo, attraverso il fatto che la parola:

$$R = C \oplus S$$

è zero.

Benché questo procedimento possa essere molto efficiente su un calcolatore digitale, si può apportare un ulteriore miglioramento. E qui entra in gioco la mia seconda idea. Vogliamo esplorare un insieme «rettangolare» di soluzioni possibili:

$$L[i] < C[i] < H\,[i],\ i = 0..15$$

dove i limiti, $L[\,]$ e $H[\,]$, debbono essere prefissati (per esempio, mediante un'analisi statistica).

Effettuiamo la ricerca nello spazio delle soluzioni possibili utilizzando quella che potremmo chiamare esplorazione «a tergicristallo», illustrata dall'esempio...

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	4	4	4	4	4	4	4	4	5	5	5	5	5	5
2	2	2	3	3	3	4	4	4	4	4	4	3	3	3
6	7	8	8	7	6	6	7	8	8	7	6	6	7	8

dove un'entità a tre componenti esplora lo spazio:

$$(4..7) \times (2..4) \times (6..8).$$

Poi a ogni nuovo controllo viene cambiata una sola delle componenti in C, rispetto al controllo precedente, e questa componente viene o aumentata o diminuita di un'unità.

Inventiamo una notazione per la parola che abbia il bit numero i fissato a 1 e tutti gli altri a zero. Chiamiamola:

$$E^i$$

Se la componente in C, modificata in una particolare iterazione, è il numero i e passa da j a $j+1$, allora si verificano, nei parametri del nostro calcolo, le seguenti modificazioni:

$$C \leftarrow C \oplus E^i$$
$$S \leftarrow S \oplus P[j]_i + P[j+1]$$
$$R \leftarrow R \oplus (E^i + P[j] + P[j+1])$$

Un'equazione analoga (con $j-1$ al posto di $j+1$) è valida quando la componente i-esima viene diminuita.

L'espressione fra parentesi può essere calcolata quando viene generato il programma. Questo vuol dire che, se si conserva in un registro il valore attuale di R, ciascun controllo di una possibile soluzione consiste in un'operazione di OR esclusivo fra il registro e una costante immediata e nel successivo confronto del risultato con zero. Ovviamente bisogna dedicare qualche istruzione alla realizzazione dell'esplorazione a tergicristallo, ma complessivamente non saranno più di una per controllo.»

Robert Wolfson:

«Consideriamo la sua [di Sallows] stima relativa al tempo che sarebbe stato necessario al suo algoritmo originale per raggiungere una soluzione. Nell'articolo a pagina 137 si parla di 31,7 milioni di anni. Evidentemente si arriva a questo valore ipotizzando di esaminare un intervallo di 10 valori per ciascuna lettera variabile nel programma, e ipotizzando che sia possibile controllare 10 combinazioni di 16 valori al secondo. Abbiamo allora:

31709791.98 ↔ 1E16 ÷ × / 10 60 60 24 365 (in notazione APL)

Ma c'è una grande ridondanza di calcoli, perché in realtà non vi sono 1E16 combinazioni diverse da controllare. Consideriamo che l'enumerazione affermata in un pangramma è vera per qualunque riorganizzazione del pangramma. Ne segue che, se una data associazione di 16 valori alle 16 lettere variabili non dà luogo a un pangramma, lo stesso accade per qualunque altra associazione dello stesso insieme di valori. Un altro modo per vederlo sta nell'osservare che le righe della matrice di valutazione del pangramma sono indipendenti fra loro: ogni qualvolta si arriva a una somma delle colonne che, come insieme, corrisponde alla combinazione del momento, si ha un insieme di valori da cui si può costruire un pangramma riflessivamente vero.

Possiamo identificare univocamente un insieme di 16 valori semplicemente elencando i numeri in ordine crescente. Se dovessimo prendere ciascuna delle 1E16 selezioni di un valore da ciascun intervallo, per poi metterle in corrispondenza in ordine crescente quanti insiemi distinti risulterebbero? Per molti che siano, questo è il numero degli insiemi di valori che dovremmo controllare per la pangrammabilità, e chiaramente è inferiore a 1E16, se c'è qualche sovrapposizione fra gli intervalli.

La valutazione di questo numero diventa più facile se si deriva un altro insieme, più semplice, di intervalli, dagli intervalli scelti origina-

riamente, quali che fossero. Ricordate che non dobbiamo più associare i valori alle lettere, finché non abbiamo stabilito che quei valori possono dare un pangramma. Ne segue che i nostri intervalli non debbono essere associati nemmeno con le singole lettere. Pertanto, possiamo ridisporre gli intervalli in modo che i loro limiti inferiori siano in ordine crescente. Poi, senza che con questo si perda in generalità, possiamo effettuare uno scambio dei loro limiti superiori, per formare un insieme di 16 coppie, i cui limiti, sia inferiori, sia superiori, formano successioni crescenti.

Ora possiamo cominciare a scegliere insiemi di 16 valori da valutare, per stabilire se soddisfano la definizione di pangramma. Semplicemente, scegliamo numeri dagli intervalli in ordine, scegliendo a ogni intervallo un numero maggiore o uguale al valore scelto dall'intervallo precedente. Ancora una volta, in quanti modi possiamo procedere?

Beh, immaginiamo di essere approdati a un numero in un certo intervallo. Chiediamoci in quanti modi possiamo completare una successione crescente di 16 valori, se il numero appena scelto dall'intervallo precedente non era maggiore del numero a cui siamo appena approdati. Potremmo:

1. Scegliere il numero a cui siamo approdati, per poi giungere al minimo numero dell'intervallo successivo, che non sia maggiore di questo, e porci nuovamente la stessa domanda; oppure
2. Andare ad approdare sul numero immediatamente più elevato, nell'intervallo in cui già ci troviamo, e porci nuovamente la stessa domanda.

Se potessimo rispondere alle domande poste in ambedue questi punti, la loro somma darebbe la risposta alla domanda iniziale.

Pertanto, possiamo seguire una impostazione «da triangolo di Pascal» per rispondere alla nostra domanda, partendo con l'elemento più elevato nell'ultimo intervallo, per procedere poi a ritroso verso l'elemento più basso nel primo intervallo.

Per esempio, supponiamo di avere solo 3 intervalli, [1,4], [3,8] e [7,9]. Possiamo costruire una tabellina in cui ciascun valore si ottiene sommando il valore che si trova immediatamente al di sopra di esso e quello che si trova alla sua destra. Nella prima riga si mette un 1 iniziale, come nel triangolo di Pascal. Ciascuna delle righe successive corrisponde a un intervallo. In una riga, le posizioni sono riempite soltanto nelle colonne che corrispondono ai valori dell'intervallo. Quando al di sopra di una posizione data manca un valore, si usa, per questo componente della somma, il valore più vicino a sinistra della riga precedente. Quando manca un valore a destra, si usa lo zero.

	1	2	3	4	5	6	7	8	9	inizio
inizio										1
[7,9]							3	2	1	
[3,8]			17	14	11	8	5	2		
[1,4]	65	48	31	14						

Il valore in basso a sinistra dà il numero degli insiemi distinti che possono essere scelti dagli intervalli specificati.

La programmazione di questo algoritmo è semplice, e con un programma del genere possiamo studiare la quantità di tempo effettivamente necessaria per una ricerca di pangrammi. Io ho scritto il programma e l'ho valutato per un insieme di intervalli, ottenuti prendendo i numeri che sono soluzione del primo pangramma nell'articolo a pagina 137, sommando e sottraendo 5 da ciascun valore. (Faccio in modo che le sottrazioni diano un valore minimo di 1.) Gli intervalli derivati sono: [1,8] [1,8] [1,8] [1,9] [1,9] [1,10] [1,11] [3,13] [3,13] [4,14] [7,17] [7,17] [14,23] [15,25] [21,31] e [24,34]. Facendo girare l'algoritmo su questi intervalli, trovo che gli insiemi da valutare sono circa 6.418E10 e, alla velocità di 10 al secondo, risultano solo 208,5 anni, molti meno di 31,7 milioni!

Sull'Amdahl V7C della mia società, con un programma APL in ciclo (che è inefficiente, a causa delle chiamate ripetute all'interprete), riesco a raggiungere i 750 controlli al secondo. Con un linguaggio adatto dovrei poter fare significativamente meglio, superare sicuramente i 1000 controlli al secondo. Con un supercalcolatore dovrei poter arrivare oltre i 10 000 controlli al secondo, e di conseguenza dovrei poter effettuare una ricerca completa su tutti questi intervalli in un paio di mesi.

Nella discussione, poi, ho considerato solo l'impostazione della ricerca esaustiva, fatta di forza bruta, senza alcuna eleganza. Poiché però fra le frequenze delle varie lettere sussistono relazioni ben definite (per esempio, vi sono almeno tante i quante x), si dovrebbe poter eliminare un numero di insiemi molto maggiore di quello che fin qui ho eliminato io.»

Letture consigliate

Per chi volesse approfondire la conoscenza dell'intelligenza artificiale, sono apparsi in lingua italiana:

B. Raphael, *Il computer che pensa*, Muzzio, Padova 1986;
E. Rich, *Intelligenza artificiale*, McGraw-Hill, Milano 1986

ambedue introduttivi ma di ampio respiro. Per chi conosce l'inglese, è consigliabile anche:

P. H. Winston, *Artificial Intelligence*, Addison-Wesley, Reading, Mass. 1984

seconda edizione, profondamente rinnovata, di un testo pubblicato originariamente nel 1977.

J. Haugeland, *Artificial Intelligence. The Very Idea*, MIT Press, Boston 1986 (traduzione italiana in preparazione presso l'editore Boringhieri)

è un testo sui fondamenti concettuali dell'intelligenza artificiale e sulla legittimità di questo tipo di ricerca, scritto in modo molto lucido e convincente.

P. McCorduck, *Storia dell'intelligenza artificiale*, Muzzio, Padova 1987

presenta materiale interessante, raccolto con gusto giornalistico dalla McCorduck con un paziente lavoro di ricerca e con interviste di prima mano ai maggiori protagonisti della ricerca in intelligenza artificiale. (Come il già citato volume di Raphael, anche questo appartiene a una collana che l'editore Muzzio ha dedicato a «Intelligenza artificiale e robotica» e che conta già una decina di volumi.)

D. Hofstadter, *Gödel, Escher, Bach*, Adelphi, Milano 1984

sfugge a qualunque classificazione, ma può essere visto come una (molto personale) introduzione all'intelligenza e alla creatività artificiale: non è un libro facile e si può essere in disaccordo con molte tesi sostenute dall'autore, ma è una lettura molto proficua.

Pregevolissimi, in generale, anche tutti gli articoli pubblicati da Hofstadter su «Scientific American» (e tradotti in italiano su «Le Scienze» a partire dal numero di settembre 1982) nella rubrica «Temi metamagici», che spesso riprendono argomenti o spunti di *Gödel, Escher, Bach*.
Consigliabile è anche:

Intelligenza artificiale, «Le Scienze quaderni», n. 25, a cura di Giuseppe O. Longo

che raccoglie numerosi articoli pubblicati da «Le Scienze» sull'argomento, insieme ad alcuni materiali originali.

Per chi è interessato soprattutto ai giochi, sono indispensabili le raccolte di Martin Gardner, pubblicate in italiano in parte dall'editore Sansoni e in parte da Zanichelli, con vari titoli; raccolgono gli articoli della rubrica «Giochi matematici» che Gardner ha tenuto per oltre vent'anni su «Scientific American» (dal 1968 tradotti anche sulle pagine di «Le Scienze»), tutti pregevoli, miglior esempio di «matematica ricreativa» del nostro secolo.

Con un impianto più teorico e sistematico è invece:

E. R. Berlekamp, J. H. Conway, R. K. Guy, *Winning Ways for your Mathematical Plays*, Academic Press, Londra 1982

in due volumi di quasi mille pagine complessive, un testo unico nel suo genere.

E. Solomon, *Programmare con i giochi*, Boringhieri, Torino 1987

è dedicato specificamente alle tecniche di programmazione di giochi (di giochi di simulazione in particolar modo) al calcolatore.

Molto semplice, infine, per cominciare a esplorare i temi dell'intelligenza artificiale con il proprio personal computer è:

J. Krutch, *Esperimenti di intelligenza artificiale in Basic*, Muzzio, Padova 1985.

Finito di stampare
nel mese di settembre 1987
dalla SAGDOS - Officine Grafiche e Legatoria
Brugherio (Milano)

Testi di A.K. Dewdney e Brian Hayes

Un programma che gioca a dama è spesso avanti di un passo

Yin e yang: ricorsività e iterazione; la Torre di Hanoi e gli Anelli cinesi

Il re (un programma per gli scacchi) è morto. Viva il re (una macchina per gli scacchi)!

Guerra dei nuclei: battaglie tra programmi a colpi di bit

Un bestiario di virus, bachi e altre insidie
per la memoria dei calcolatori nella Guerra dei nuclei

Il programma MICE *si fa strada fino alla vittoria nel primo torneo di Guerra dei nuclei*

Un sistema esperto batte i semplici mortali nella conquista dei Sotterranei del Destino

Una tortuosa odissea da Robotropoli alle porte elettroniche di Silicon Valley

Star Trek emerge dalla clandestinità e trova il suo posto fra i videogiochi domestici

Squali e altri pesci combattono una guerra ecologica sul pianeta Wa-tor

Un calcolatore usato come telescopio per incontri ravvicinati con ammassi stellari

Un sublime volo di fantasia su una base di dati deserta

Dove si introduce il lettore ai piaceri del calcolo

Dove si parla dell'automa finito:
un modello minimale delle trappole per topi, dei ribosomi e dell'anima umana

L'automa cellulare offre un modello del mondo e un mondo in se stesso

Un calcolatore trappola per l'alacre castoro, la più attiva fra le macchine di Turing

Costruire calcolatori a una sola dimensione getta luce
su fenomeni irriducibilmente complessi

Guardando la geometria dal di dentro, a passeggio con una tartaruga

Un microscopio al calcolatore per gettare uno sguardo
sul più complesso fra gli oggetti della matematica

Montagne frattali, piante graftali e altra grafica al calcolatore della Pixar

Ai Laboratori Bell il lavoro è gioco e le malattie dei terminali sono benigne

Tappezzeria per la mente: immagini al calcolatore quasi, ma non del tutto, ripetitive

Caricature al calcolatore e strani viaggi nello spazio dei volti

Un rapporto di ricerca sulla sottile arte del trasformare letteratura in non senso

Un giardino informatico in cui germogliano anagrammi, pangrammi e qualche erbaccia

Pazzia artificiale: quando un programma schizofrenico
incontra un analista computerizzato

In appendice:

Linee guida per la Guerra dei nuclei

Linee guida per il cannone ad alianti

Algoritmi realizzati e proposti per la soluzione del problema
del pangramma di Lee Sallows

L. 17.000
(16.667)

627446 E